SCALING ROBOT ADAPTATION WITH LARGE MODEL GUIDANCE

by

Jesse Zhang

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2025

Acknowledgements

Give no decision till both sides thou'st heard

Phocylides

In my case, it's till *three sides I've heard.* Thank you to my advisors, Erdem Bıyık, Joseph Lim, and Jesse Thomason for their mentorship. I am grateful to Joseph for teaching me how to think about research problems and encouraging me to constantly set higher expectations for myself, and being critical when needed to help me learn. I would also like to thank him for continuing to offer financial support for academic travel despite all the complications of doing so from another academic institution in a different country. I am thankful to Jesse for welcoming me into his lab when I was stuck in PhD student limbo at USC and unsure of what to do, for helping me think of problems from a different, well-grounded perspective, and offering extremely detailed feedback when I need it. I am also thankful to Erdem for offering to take me as an extra unexpected student in his first year as a professor, for always being available for any question, and being open to advising on any research direction that I am interested in. All three of my advisors have been immensely helpful in different ways, and I am extremely happy to have gone through this unique experience of being able to ask for feedback from three diverse perspectives.

Secondly, thank you to the other members of the committee, Feifei Qian and Gaurav Sukhatme, for lending their precious time to me. I specifically appreciate Feifei's feedback during my thesis defense asking me to also think about how to apply the methods presented in this thesis to other sub-fields in robotics. I also thank Gaurav for additionally being on my quals committee and asking critical, insightful questions

during that time with respect to how to place my work in the general context of robotics that I still think about to this day. I also appreciate the Viterbi School of Engineering for offering financial support through an additional fellowship during the process of switching advisors, along with all the helpful people in the administration who helped along the way: Andy Chen, Lizsl De Leon, Ellecia Williams, and Asiroh Cham.

Importantly, I would like to thank my family: my parents, Lin Zhang and Hong Luo, have been incredibly supportive throughout my PhD even though I did not become a *real doctor*; my big brother Kenny, for always believing in me and being in close proximity through most of my PhD, and my partner Elisabeth for moving down to LA for me and constantly inspiring me. My grandpa is also a great source of inspiration, still doing research in his 90s, and I always enjoyed calling him to tell each other what we were working on.

I am very glad to have become great friends with an incredible group of labmates across three labs at USC. Special thanks to Karl Pertsch, Youngwoon Lee, and Shao-Hua Sun from Joseph Lim's lab for their incredible guidance during the first few years of my PhD. I especially thank Karl as he and I collaborated from nearly the start of my PhD all the way to year three. I learned about how to think about research, how to write papers, and how to overcome difficult technical challenges through all three of them. I also had an incredible set of peers and collaborators with whom I became great friends: Ayush Jain, Grace Zhang, Jiahui Zhang, Anthony Liang, Abrar Anwar, Sumedh Sontakke, Sid Devic, Tejas Srinivasan, Ishika Singh, Yutai Zhou, Yigit Korkmaz, Pavel Czempin, Minho Heo, Lee Kezar, Yusen Luo, Xinhu Li, Ziyi Liu, Ryan Lindeborg, Bingjie Tang, and many others. I especially thank Sumedh, Anthony, Tejas, Yutai, Abrar, and Sid for being great lifting, (sometimes) tennis, (only occasionally) Valorant, or running buddies. I additionally thank Sungjae Park, Lucy Shi, Shubham Sharma, Haeone Lee, along with everyone who was in any of my three labs for great discussions, feedback, and brainstorming throughout my PhD. For nearly every paper introduction I wrote, Laura Smith and Sid Kaushik looked over the draft and gave me great feedback, I thank both of them for doing so.

Finally, throughout my PhD, I got to collaborate with great mentors whom I met during internships. I especially thank Minsuk Chang, Abhishek Gupta, Anqi Li, Fabio Ramos, Haonan Yu, and Rasool Fakoor for being great mentors at NAVER, NVIDIA, Horizon Robotics, and AWS during my PhD.

Table of Contents

List of Tables

List of Figures

Abstract

General-purpose robots deployed in the real world must respond to dynamic environments and continuously learn new tasks. However, existing methods struggle to support such *adaptation at scale*—that is, without substantial human supervision. My thesis presents an approach to scalable robot adaptation by leveraging the general knowledge encoded in Large Pre-trained Models (LPTMs). I show how integrating LPTMs with robot learning frameworks can: (1) enhance robot pre-training to better prepare for unfamiliar tasks and settings, (2) adapt to new tasks and environments with human feedback, and (3) ultimately enable autonomous adaptation with minimal human input. Together, these contributions outline a path toward generalizable algorithms that empower robots to learn novel tasks in real-world, unstructured environments.

Introduction

Chapter 1

Introduction

When I began my PhD in August 2020, I wanted to tackle one central problem: *"How can we train robots that continually learn new skills and adapt to new environments after deployment?"* At the time, robot learning approaches fell into two buckets: (1) imitation learning, which relies on human-collected expert demonstrations for every new task, and (2) reinforcement learning from scratch, which is too slow and sample-inefficient. Both approaches were impractical to scale to robots that can continually adapt after deployment.

Meanwhile, we humans are remarkably capable learners who can continually learn new skills throughout our lifetimes and fluidly transfer expertise across domains. For example, a world-class cardiothoracic surgeon with the ability to carefully perform precise, fine-grained cuts around patients' hearts can go home after work and apply her fine-grained hand control to learning how to play the piano.

Initially, I wasn't sure how to bridge the gap between how humans learn and how robots *could*. After all, the problem setting that I cared about, robots adapting to new tasks without extensive retraining, has been studied for decades [25, 124]—what more could I contribute? My guiding principle at the time was that **humans have strong priors**, both evolutionarily and gained through experience, that help with the ability to learn new skills [102, 12]. But where do we get these priors for robotics?

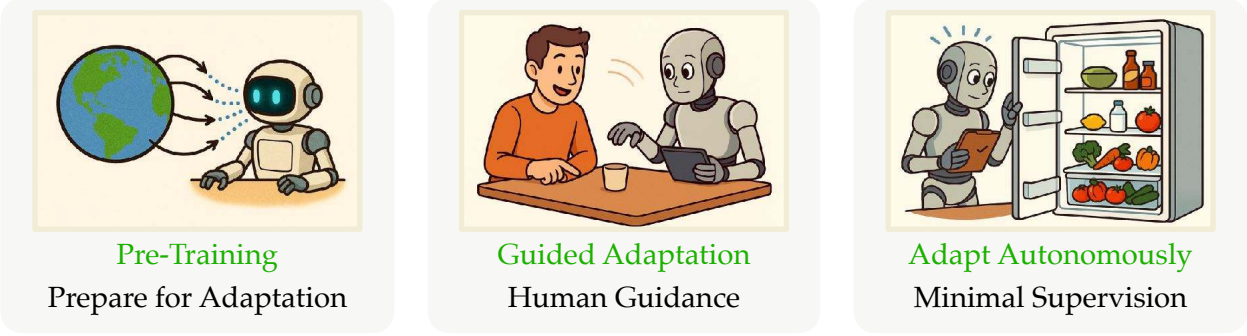


Figure 1.1: The core components of **scaling robot adaptation with large model guidance**. I introduce how to use large pre-trained models (LPTMs) to first guide robot **pre-training** so they can see as much data as possible before adapting to new tasks or settings. Then, I demonstrate how to use LPTMs to enable **guided adaptation** by using them to interpret human guidance during adaptation. Finally, I present how to use LPTMs to help robots **adapt autonomously**, with minimal human supervision, to new tasks and settings.

I did not truly know how to answer this question, and during the early years of my PhD, I worked on robotics research problems related to but not directly tackling this challenge [406, 354, 403]. In early 2022, I finally came back to thinking about priors for enabling robot learning. By 2022, large pre-trained models (LPTMs) such as GPT-3 [42], trained on terabytes of human-generated internet text, were starting to become mainstream. They were demonstrating promise in serving as *knowledge priors*, demonstrating potential to be useful for a variety of downstream tasks in the fields of natural language processing and computer vision [290, 42, 364, 144]. This insight became the turning point of my research direction. I began to explore how LPTMs could serve as prior knowledge sources for robots—enabling adaptation by helping robots to pre-train on more data, interpreting human guidance for robots, or even guiding robots in what new tasks to learn once deployed.

This thesis explores how to leverage large pre-trained models (LTPMs) to enable **scalable robot adaptation**. This thesis focuses on using LPTMs to help both robot *pre-training* to encourage learning strong robotics priors before adaptation, and *adapting* to new tasks after deployment. I segment this thesis into three parts:

1. **Pre-training** robots to learn strong priors that help with learning new tasks or in scenes;

2. After deployment, **adapting with human guidance** to new scenes and tasks; and finally

3. **Adapting with minimal human supervision** as a step towards truly autonomous robot learning.

See Figure 1.1 for an overview of my thesis.

1.1 Pre-training Robot Policies for Efficient Adaptation

In Part I, I introduce three approaches aimed at pre-training robots *before deployment* to prepare them for adaptation after deployment. This chapter is driven by one key idea: strong pre-training algorithms should train robots to **maximally use offline data** to learn as many tasks across as many scenes as possible, so that they can bootstrap this knowledge for adapting to new tasks and scenes once deployed. Furthermore, to enable *scalability*, pre-training should require as little manual human annotation as possible. To this end, I include three chapters on pre-training. From here on, I will be using "we" to describe each individual chapter as they could not be possible with a great set of collaborators.

In Chapter 3, we introduce SPRINT, a method to automatically expand an existing dataset of *language-annotated trajectories* by over 2.5X its original size, greatly reducing human annotation effort through two novel procedures that combine LPTMs and offline reinforcement learning (RL): *Aggregating* language annotations and employing large language models (LLMs) for instruction relabeling (e.g., *put mug in coffee machine* + *press brew* $\rightarrow$ make coffee), and *Chaining* skills from *different* trajectories through a principled offline RL objective, helping robots prepare for learning new tasks by teaching them to stitch behaviors together in a manner not represented explicitly in the data. Through these procedures, we achieved up to 8x improvement in zero-shot performance and more efficient online RL adaptation to novel tasks, across both simulated [328] and real-world benchmarks [76]. SPRINT was published in Zhang et al. [405]. However, while SPRINT addressed how to expand an existing dataset of language annotated trajectories, what if we there are no annotations to begin with?

We answer this question in Chapter 4, where we present EXTRACT, a pre-training method that uses large pre-trained vision language models (VLMs) to *autonomously* extract a discrete set of parameterized, task-agnostic skills which can act like functional robot API calls (e.g., `pickup(x=0.45, y=0.5)`) from offline data. Naïvely applying VLMs directly did not work well. Instead, we used VLM image encoders to produce *embedding differences* that encode high-level behaviors across parts of each trajectory video. Then, we clustered these differences into a discrete set of behaviorally aligned skills that a robot policy learn a small set of continuous arguments for to quickly adapt to a new task. Our experiments demonstrate up to a 10x sample efficiency improvement over prior skill-based RL methods [282] in learning *new tasks* across multiple benchmarks [104, 201], including a real-world Furniture Assembly task [135] that improved 32% with just 100 trajectories of real-world online RL fine-tuning. This work was published as Zhang et al. [402]. Both SPRINT and EXTRACT focused on learning as much as possible from robotics data. But, it would be even better if we could pre-train with *arbitrary internet data*, of which there is far more than robotics-specific data, to further help with generalization to new tasks.

Chapter 5 enables pre-training with arbitrary internet data by introducing HAMSTER, a large, *hierarchical* vision-language-action model where the top-level of the hierarchy is pre-trained on internet data to learn *robotics*-related tasks without requiring as much labeled robotics data. In HAMSTER, we propose to fine-tune a VLM to use 2-dimensional *paths*, easy to obtain at scale from simulated robotics data and existing open-source robotics datasets, to predict high-level robotics actions in the form of these paths. This VLM can then be used zero-shot for providing high-level path guidance to a robot policy that requires less fine-tuning data to adapt than prior vision-language-action models [165]. This work was published as Li et al. [192].

These first 3 chapters discuss how to use LPTMs to help scalably pre-train robot policies. In the next section, we describe one easy way to adapt these policies to new scenes and tasks: via some form of human guidance.

1.2 Adapting to New Scenes and Tasks with Human Guidance

Directly providing human guidance is a natural way to help robots adapt, but doing so in a scalable manner is important to be able to actually deploy robots. In Part II, I introduce two approaches that aim to minimize the amount of human guidance required to adapt robots to new tasks and scenes.

In Chapter 6, we train robots to *continually adapt* to new tasks and scenes with a few human demonstrations as guidance per task or scene. When this project was being worked on, there was little focus on how to continually adapt large architectures derived from LPTMs on new demonstration data. We found that fine-tuning a model with so many parameters on small datasets performs poorly, yet assuming access to ginormous per-task datasets is often unreasonable for consumer-facing robots. Therefore, we introduce TAIL, a method that incorporates low-rank adapters common in other fields [285, 55, 284] with robotics policies derived from large pre-trained model backbones [290, 289]. We found that with LoRA [142], we were able to adapt large robot policies with just ∼1% of the trainable parameters of the original model while avoiding catastrophic forgetting. Nowadays, LoRA-finetuning is common across all large pre-trained robotics models [165, 32]. This work is published as Liu et al. [210]. Still, assuming per-task human demonstration datasets is still not very easy to scale. Is there another way to provide human guidance more easily?

In Chapter 7 we introduce HAND, a simple and time-efficient method for teaching robots new manipulation tasks through *human hand demonstrations*. Rather than provide human guidance in the form of per-task demonstrations, a human provides guidance once before deployment by providing easy-to-collect robot *play data*. Then, using a visual tracking pipeline powered by a vision LTPM, HAND extracts the motion of the human hand from the hand demonstration and retrieves robot sub-trajectories in two stages: first filtering by visual similarity, then retrieving trajectories with similar behaviors to the hand. Fine-tuning a policy on the retrieved data enables real-time learning of tasks in under three minutes, without requiring calibrated cameras or detailed hand pose estimation. Experiments in simulation [237] and

on real robots also show that HAND outperforms retrieval baselines [197, 238] by over 2× in task success rates. This paper has been submitted to a conference and is under review.

These chapters demonstrated ways to adapt with some form of human guidance. While they are more scalable than traditional IL methods, a question remains: Can we enable robots to *autonomously adapt*, with *minimal human supervision*? This is the question I investigate in the last section.

1.3 Scalable Adaptation with Minimal Human Supervision

In Part III, I investigate ways to enable scalable robot adaptation while reducing the amount of supervision required. This part represents a step towards truly human-like *autonomous* robots that acquire new skills on their own. To this end, I discuss three chapters.

In Chapter 8, I investigate how to enable robots to learn complex tasks in *new environments* without explicit human task guidance. Our method, BOSS, tackles this by proposing an LLM-guided "practice" phase, where the robot refines pre-trained skills and composes them into new behaviors—without human task guidance. After pre-training, the robot is deployed in an unseen environment and autonomously selects skills to master and compose. For example, after mastering *pick up empty coffee mug*, an LLM might suggest *put mug in coffee machine* from its skill library. The robot then attempts this new skill sequence and integrates it as a new skill, *make coffee*. As the robot practices, its skill repertoire—tailored specifically to that environment—grows richer. This method, published as Zhang et al. [407], enabled learning new skills in new environments with just 17,000 environment steps *in the real world*. However, due to the difficulty of hand-writing dense reward functions, we used sparse success detection to reward the robot's practice. Dense rewards are much more effective in teaching a policy to learn difficult tasks [340]. In the last two chapters, I investigate approaches for replacing human guidance in providing dense reward functions by using LPTMs.

Chapter 9 describes my first attempt at LPTM-guided dense rewards, RoboCLIP. In this work, we used pre-trained VLMs to generate rewards for new tasks based solely on either a single demonstration video or language instruction. RL agents trained with RoboCLIP rewards demonstrate 2-3 times higher zero-shot performance than competing imitation learning methods on downstream robot manipulation tasks, doing so using only one video demonstration or language instruction. RoboCLIP is published as Sontakke et al. [334]. However, we noticed that RoboCLIP performed much better with a video demonstration; the language-guided rewards seemed to be far less stable for policy training. Ideally, humans would be able to only give language commands to a robot for it to be able to learn that task as a video demonstration still requires signinficant human effort to collect that demonstration through robot teleoperation. This problem leads me to the final chapter of my thesis.

Chapter 10 proposes ReWiND, a framework for tackling the problem of sample-efficient, real-world learning of new tasks using only a language description. ReWiND starts from a small demonstration dataset to learn: (1) a data-efficient, language-conditioned reward function that labels the dataset with rewards, and (2) a language-conditioned policy pre-trained with offline RL using these rewards. Given an unseen task variation, ReWiND fine-tunes the pre-trained policy using the learned reward function, requiring minimal online interaction. We show that ReWiND's reward model generalizes effectively to unseen tasks, outperforming baselines by up to 2.4X in reward generalization and policy alignment metrics. Finally, we demonstrate that ReWiND enables sample-efficient adaptation to new tasks in both simulation and on a real bimanual manipulation platform, taking a step towards scalable, real-world robot learning. This work has been submitted to a conference and is under review.

In the final chapter, Chapter 11, I address remaining unsolved problems that prevent truly autonomous robots from being deployed. These are problems I hope to tackle after my PhD as a continue my research journey.

Chapter 2

Background

The techniques to train robots in this thesis are all variants of reinforcement learning (RL), imitation learning (IL), and offline reinforcement learning approaches. I first define these techniques and describe their use cases and notation used in the rest of the thesis chapters.

2.1 Reinforcement Learning (RL)

RL describes a class of methods that are aimed at solving arbitrary sequential decision making tasks. In RL, there is an agent, such as a robot, that interacts with an environment. This environment is typically defined as a discrete-time, finite-horizon Markov decision process (MDP) described as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, T, R, \mu, \gamma)$. $\mathcal{S}$ denote the state space where each $s \in \mathcal{S}$ denotes the full state of the world, such as a robot's exact pose and all poses and velocities of all relevant objects for the MDP. $\mathcal{A}$ denotes the action space, such as robot arm movements and gripper open/close actions. The transition distribution $\mathcal{T}$ denotes how states change from time t to $t+1$ as the robot takes actions, i.e.,

$$s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t). \tag{2.1}$$

The agent starts in an initial state sampled from $s_0 \sim \mu(s)$ where μ represents the (possibly unknown) starting state dsitribution. The total number of timesteps the agent can take before the environment resets itself according to μ is T.

At each timestep, the agent receives a reward r_t from the reward function $R(s_t, a_t, s_{t+1})$ that gives feedback regarding how good or bad the action taken was in the MDP. For example, the robot may receive positive reward for picking up an object in a pick-place MDP, and negative reward for colliding with the table. In this thesis, we denote the agent's action distribution as $\pi(a \mid s)$, where the goal of the agent is to search for an action distribution π that maximizes the expected cumulative reward over the entire episode discounted by the final term in the MDP-tuple, γ:

$$\max_{\pi} \mathbb{E}_{\pi,\mu,\mathcal{T}} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}) \right]. \tag{2.2}$$

In robotics, we commonly consider Partially Observed MDPs, where we do not necessarily know full state information $\mathcal{S}$, but instead the agent observes *observations* o_t from an observation space $\mathcal{O}$ that only include partial information. Examples of observations can be RGB camera image observations from a robot's front camera. Many works in the literature use $\mathcal{S}$ and $\mathcal{O}$ interchangeably. In this thesis, I use observations o whenever I am assuming specific observation types, such as RGB camera observations. In other cases where no specific structure of the observation space is assumed, I use the state notation s so that notation is more similar to most works in RL. Finally, some of my chapters, especially in Part III, do not assume access to the reward function and instead learn the reward function R. However, the overall goal of the policy remains the same as in Equation (2.2), just that the agent may be optimizing π over a learned reward function instead of one given by the MDP.

2.2 Imitation Learning (IL)

Another approach to training robots that I use in Chapter 5, Chapter 6, and Chapter 7 is *imitation learning*, where the agent trains π to mimic demonstration data in a pre-collected dataset $\mathcal{D}$.

In a standard imitation learning setting, there is no reward function. Instead, the goal is to maximize the probability of mimicing ground truth actions sampled from the dataset $\mathcal{D}$ given the same states. Therefore, the imitation learning objective for a probabilistic policy $\pi(\cdot \mid s)$ can be described as:

$$\max_{\pi} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\log \pi(a \mid s)\right], \tag{2.3}$$

where we are maximizing the log of the probability of the given action (equivalent to maximizing the probability of the action).

2.3 Offline Reinforcement Learning

Finally, one approach that I use throughout the entire thesis is offline RL, also occasionally referred to as *batch RL* in the literature. In Offline RL, we also assume access to a pre-collected dataset $\mathcal{D}$ just like in IL, but we typically also know the reward function. However, the agent's goal is still to maximize the same objective as in RL for a given MDP (Equation (2.2)). Therefore, offline RL approaches typically train the policy on some objective that trades off maximizing the *rewards* in the dataset $\mathcal{D}$ while constraining the policy to the *actions* in $\mathcal{D}$, essentially blending Equation (2.3) and Equation (2.2) in different ways depending on the specific algorithm.

In this thesis, as I am focused specifically on *adaptation*, offline RL is used in the context of first pre-training offline using a given offline RL algorithm and then taking the learned policy π online to continue maximizing rewards in the MDP that generated the data in $\mathcal{D}$ or to maximize rewards for a separate, but related, MDP that represents a new task or new scene for the robot to adapt to.

Part I

Pre-training Robots Policies for Efficient Adaptation

Chapter 3

Scalable Policy Pre-Training via Language Instruction Relabeling

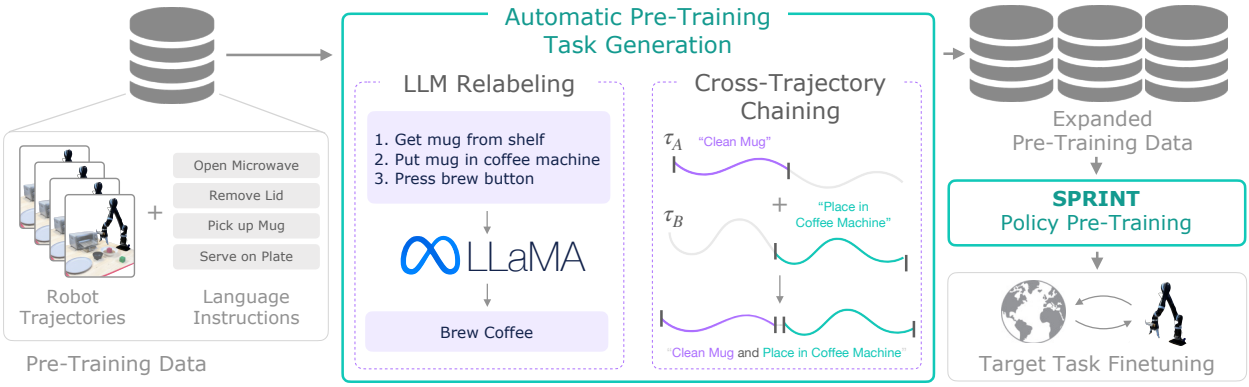


Figure 3.1: SPRINT is a scalable approach for pre-training robot policies with a rich repertoire of skills while minimizing human annotation effort. Given a dataset of language-annotated trajectories for offline pre-training, SPRINT automatically expands the skill set via LLM-based **instruction relabeling** and **cross-trajectory skill chaining** to enable efficient finetuning on unseen target tasks.

3.1 Introduction

When humans learn a new task, e.g., how to cook a new dish, we rely on a large repertoire of previously learned *skills*, like "*chopping vegetables*" or "*boiling pasta*", that make learning more efficient. Similarly, much work in robot learning aims to equip robots with a set of useful skills for improving learning efficiency [342, 305, 131, 217, 282, 121]. A common approach to acquiring a rich skill set is to pre-train policies on a wide range of tasks. Recent works have employed *language instructions* as a way for humans to manually define such tasks for policy training, typically via hindsight annotation of large, pre-collected

robot experience datasets [237, 218, 219, 39]. While the resulting policies show impressive capabilities, generalization to new tasks requires a *large* set of pre-trained skills and thus many pre-training tasks. As a result, prior works resorted to annotating robot trajectory datasets with *hundreds of thousands* of human instruction labels [219], limiting their application outside industrial contexts. Can we instead devise a pre-training approach that similarly equips robots with a wide repertoire of skills but *minimizes* the need for human task annotations?

We introduce SPRINT (**S**calable **P**re-training via **R**elabeling Language **IN**s**T**ructions), a scalable pre-training approach that equips robots with a large set of skills while substantially reducing human labeling effort (see Figure 3.1). Given an initial set of language-labeled pre-training tasks, SPRINT uses extensive *automated* relabeling to greatly expand this task set without additional human effort. Given a dataset of robot trajectories with initial language instruction annotations, we leverage two core ideas to grow the number of tasks. First, we leverage the rich knowledge captured in large language models (LLMs) to iteratively combine consecutive language instructions into more complex tasks, e.g., "*place mug in coffee machine*" and "*press brew button*" into "*make coffee*". Second, we propose a language-conditioned offline reinforcement learning (RL) objective that "stitches" multiple trajectory segments from the data to form new tasks, a process we call "skill chaining" since it allows the policy to learn longer-horizon skills. Through the combination of both techniques, SPRINT creates a richer pre-training task set that can help the agent generalize to new tasks. We demonstrate that SPRINT-pre-trained robots can leverage their resulting larger skill repertoire to more efficiently learn new downstream tasks.

In summary, our contributions are threefold: (1) we propose SPRINT, a scalable pre-training approach for robot policies that minimizes human task annotation effort via LLM-based aggregation and cross-trajectory skill chaining, (2) we introduce ALFRED-RL, an RL benchmark for the popular ALFRED household task simulator [328], to test our pre-trained agents on a rich set of long-horizon, semantically meaningful tasks, (3) we demonstrate that policies pre-trained with SPRINT learn downstream tasks more efficiently than prior pre-training approaches, both on challenging ALFRED tasks and in a real robot kitchen manipulation setup.

3.2 Related Work

Language in RL. There is a large body of work at the intersection of natural language processing and behavior learning for robotics, and the field has been further accelerated by the recent successes in training large, general-purpose language models. Language has been used to structure agents' representations [13, 252], learn reward functions [98], guide task learning via recipes [35, 14] and perform long-horizon planning [144, 7, 147, 331]. Another line of work has used language to define a wide range of tasks for pre-training policies, resulting in impressive generalization capabilities [218, 219, 39]. Yet, these works require collecting hundreds of thousands of costly human language instructions. Our approach SPRINT builds on this line of work but introduces two novel objectives for *automatic relabeling* of training task instructions, thereby substantially reducing the amount of human labeling required for successful pre-training. Prior works have also investigated automated language instruction generation [65, 63, 189], but they focus on online learning and make assumptions that are hard to scale, e.g., hand-defined grammars [65] or privileged state information [189, 63]. In contrast, we perform *offline* pre-training and use large language models for *scalable* task generation.

Pre-training Policies for RL. Developing policy pre-training approaches for faster downstream learning has been investigated for many years [148, 348, 136]. Recent advances in offline RL [187] enabled

approaches that can pre-train agents offline and effectively finetune them on online tasks [277, 330, 250, 171]. However, these approaches require target-task reward annotations on the pre-training data and the resulting policies are only pre-trained to solve the target task. Meta-RL approaches, on the other hand, pre-train on a range of tasks and thus allow fast adaptation to *unseen* downstream tasks [89, 99, 294, 256], yet require the tedious manual definition of pre-training tasks by experts. To avoid manual task design, other works have explored unsupervised pre-training approaches based on behavior diversification [2, 93, 319], extraction of behavior priors from offline agent experience [282, 9, 329] or goal state reaching [240, 49]. Closest to ours, Chebotar et al. [49] proposes an objective that randomly selects states to chain together existing trajectories, while we propose a language skill chaining objective that allows SPRINT to execute new, composite language instructions. Such unsupervised pre-training approaches [49] learn skill repertoires without clear meaning, which, as we demonstrate in Section 3.4, lead to worse downstream task transfer.

Pre-trained Models for Data Augmentation. Obtaining robot (pre-)training data at scale is costly. Thus, recent works have explored using world knowledge captured in large pre-trained models for enriching robot learning datasets, e.g., by increasing the visual diversity of trajectories [395, 56, 230] or annotating unlabeled data [374]. Our approach similarly leverages pre-trained (language) models for automated data augmentation. By investigating an orthogonal augmentation direction, aggregation and chaining of natural language instructions, SPRINT is complementary to these methods.

3.3 SPRINT: Scalable Policy Pre-Training with Language Instructions

In this paper, we propose SPRINT (**S**calable **P**re-training via **R**elabeling Language **IN**s**T**ructions), an approach for pre-training robot policies that equips them with a rich repertoire of skills to enable efficient finetuning on unseen tasks. Following prior work on agent pre-training, SPRINT assumes access to a large offline dataset $\mathcal{D}$ of agent experience [118, 217, 282, 49, 91, 283], collected, e.g., from prior RL runs or via

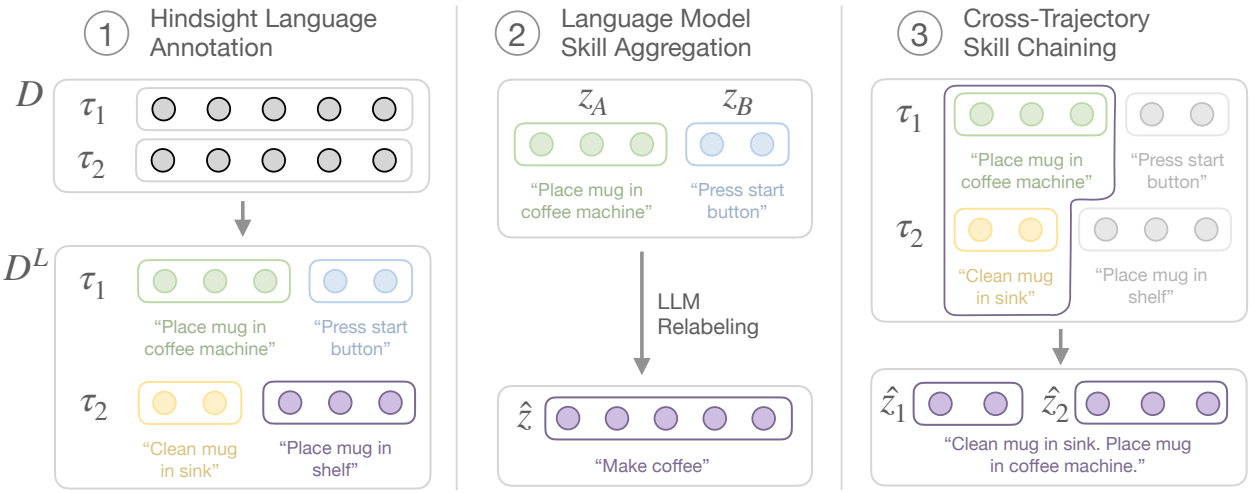


Figure 3.2: SPRINT overview. We assume access to a dataset of agent experience with language instructions for the performed skills **(1)**. Collecting such instructions with human hindsight annotation is a flexible yet costly approach for defining pre-training tasks. Thus, SPRINT introduces two approaches for automatically growing the set of pre-training tasks without additional human effort: **(2)** by aggregating language instructions with an LLM and adding the relabeled trajectories back into the pre-training dataset (Section 3.3.2), **(3)** by performing cross-trajectory chaining of skills to enable pre-training of skills that are unseen in the offline agent experience (Section 3.3.3).

teleoperation. We further assume that the data is annotated with an initial set of natural language task instructions, e.g., "*put a mug in the coffee machine*" or "*push the brew button*", that can be collected *in hindsight* via platforms like Amazon Mechanical Turk [218, 328]. Given a sequence τ of states and actions from the dataset $\mathcal{D}$, annotators can label sub-trajectories $\tau_1 = [s_0, a_0, s_1, \dots], \tau_2 = \dots$ with free-form language descriptions $z_1, z_2, \dots$ of the skills executed in the respective sub-trajectories (see Figure 3.2, left), resulting in a *language-annotated* dataset $\mathcal{D}^L$.

Approach Overview. SPRINT equips policies with a diverse repertoire of skills via language-instruction-conditioned offline RL: given a natural language task description z, the policy $\pi(a|s, z)$ is rewarded for successfully executing the instruction (Section 3.3.1). Intuitively, the richer the set of task instructions during pre-training, the more skills the policy will learn and the more downstream tasks it can finetune on efficiently. Thus, SPRINT introduces two approaches for increasing the scale and diversity of the pre-training task instructions without requiring additional costly human inputs. Firstly, SPRINT leverages

pre-trained language models to aggregate consecutive instructions into new tasks (Figure 3.2, middle, Section 3.3.2). Secondly, SPRINT introduces an objective for cross-trajectory skill-chaining via offline RL that generates novel instruction chains *across different trajectories* (Figure 3.2, right, Section 3.3.3). SPRINT pre-trains policies on the combined set of tasks and thereby equips them with a richer skill repertoire. In our experiments (Section 3.4) we demonstrate that this leads to more effective learning of new tasks.

3.3.1 Instruction-Conditioned Offline RL

To pre-train our policy π with the natural language instruction dataset $\mathcal{D}^L$, we take inspiration from goal-conditioned RL [155, 307, 49]: instead of rewarding the policy for reaching goal states, we condition our policy $\pi(a|s, z)$ on *language instructions* z from $\mathcal{D}^L$ and provide a scalable sparse reward $R(s, a, z)$ to the agent for reaching the end-state s_T of the sub-trajectory. Formally, we define the reward as:

$$R(s, a, z) = \begin{cases} 1, & \text{for } s = s_T \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

We train our policy $\pi(a|s, z)$ to maximize this reward with offline RL [187] using an instruction-conditioned critic $Q(s, a, z)$. Specifically, we use Implicit Q-Learning [171] as it is performant and easy to tune.

3.3.2 Language-Model-Based Instruction Aggregation

Large language models (LLMs), trained on massive corpora of internet text data, have been shown to be effective at performing a variety of tasks – from question answering to program synthesis – when prompted with relevant text [80, 43, 364, 293, 139, 408, 58]. Here we use LLMs to *aggregate*, i.e., paraphrase, the existing language instructions in $\mathcal{D}^L$ (see Figure 3.2, middle). Given a trajectory that contains multiple sub-trajectories, we can aggregate adjacent sub-trajectories into a longer trajectory and relabel its natural

language annotation with a summary of the individual instructions generated by the LLM, thereby generating a new *higher-level* pre-training task that encompasses instructions from multiple sub-trajectories.* We use a simple summarization prompt to instruct the language model (see Figure 3.3). Specifically, we aggregate with LLAMA-13B [352], an open-source 13 billion parameter LLM which is able to retain important information from individual instructions in the overall summary. Like in Section 3.3.1, the reward for this new aggregated sub-trajectory is 1 at the last transition and 0 otherwise. For example, we prompt the LLM to summarize the two skills (z_1 : "*Put a mug in the coffee machine*," z_2 : "*Push the brew button*"), resulting in a new annotation $\hat{z}_{1:2}$ describing both skills (e.g., "*Make coffee*"). We then add the new trajectory back to our dataset $\mathcal{D}^L$. Using this technique, we generate new language annotations for all combinations of consecutive sub-trajectories in our dataset. In practice, this increases the number of task instructions by 2.5x in ALFRED and 2x in our robot manipulation dataset (see Section 3.4).

3.3.3 Cross-Trajectory Chaining

In addition to generating new pre-training tasks composed of behaviors within the *same* trajectory (Section 3.3.2), we also want to be able to generate pre-training tasks containing behaviors across *different* trajectories. For example, if trajectory (A) shows cleaning the mug in the sink while trajectory (B) starts with placing the mug in the coffee machine, the agent should be able to learn to clean the mug in the sink and then place it in the coffee machine (see Figure 3.2, right), thus learning long-horizon behaviors that are unseen in the training data. Agents trained with standard offline RL can implicitly combine tasks described from multiple trajectories into longer-horizon behaviors via value propagation, i.e., perform "stitching" [187]. In our case of *instruction-conditioned* offline RL, values do not naturally propagate from trajectory (B) back to trajectory (A) due to the different language instruction conditionings for the critic $Q(s, a, z_A)$ and $Q(s, a, z_B)$. However, we can actively add "chaining examples" [49], which encourage

*Other relabeling operations, such as splitting an instruction into lower-level instructions, can also be performed by the LLM. However, such operations require grounding the LLM in the agent's observations to determine sub-trajectory split points. We leave investigating this to future work.

learning longer-horizon behaviors, to our training dataset by first *combining language instructions* and then appropriately *relabeling rewards*. To build such chaining examples, we first sample two sub-trajectories τ_{z_A} and τ_{z_B} from *different* trajectories (see Figure 3.2, right). Next, we create an aggregate instruction $\hat{z}$ which indicates that the agent first finishes (A) and then finishes (B), e.g., "*clean the coffee mug (A) and place it in the coffee machine (B).*"[†]

LLM Prompt Example

Summarize the following steps.

1: Pick up the tomato slice.

2: Heat it up in the microwave.

Summary: Microwave a tomato slice.

1: [SKILL 1]

2: [SKILL 2]

...

Summary:

Figure 3.3: A shortened example of the LLM prompt.

Unlike in Section 3.3.2, we cannot simply concatenate the two trajectories together and relabel the reward of the last transition to 1. Since we sampled the two sub-trajectories at random, the last state of the first, s_{T_A}, does not directly transition into the first state of the second. To solve this issue, we relabel both τ_{z_A} and τ_{z_B} with the aggregate instruction $\hat{z}$ and treat them as *separate* trajectories with appropriately labeled rewards. For transitions in τ_{z_B}, we simply relabel the last transition with a reward of 1 to be consistent with the 0-1 rewards in Sections 3.3.1 and 3.3.2. Meanwhile, we would like to relabel the reward of the last, terminal transition in τ_{z_A} so that the learned Q-value for this transition, $Q(s_{T_A}, a_{T_A}, \hat{z})$, will also be consistent with the prior labeling schemes. What reward should we use here?

[†]Note that we could generate $\hat{z}$ using the same LLM summarization as in Section 3.3.2. Yet we found the resulting summaries to often be confusing since randomly paired instructions *from different trajectories* can rarely be summarized meaningfully. We got the best empirical results by simply concatenating the sampled instructions with the word "and". Note that we perform chaining on both the original trajectories and those generated by LLM aggregation in Section 3.3.2.

Recall that Q-functions trained for sparse reward (Eq. 3.1) intuitively represent a value proportional to the probability of reaching goal state s_{T_z} at time T [49, 94]:

$$Q^\pi(s_t, a_t, z) = \mathbb{E}[\sum_{t'=t} \gamma^{t'-t} R(s_{t'}, a_{t'}, z)] = \mathbb{E}\left[\gamma^{T-t} \mathbb{1}\left[s_T = s_{T_z}\right]\right] \propto P^\pi(s_T = s_{T_z} | s_t, a_t). \tag{3.2}$$

where $\gamma \in (0, 1)$ denotes the discount factor. Following this intuition, the Q-value learned for the last transition of (A) should be proportional to the probability of finishing *the remainder* of the combined task $\hat{z}$, i.e., proportional to the likelihood of finishing (B) from s_{T_A} when taking action a_{T_A}. Following Eq. 3.2, $Q(s_{T_A}, a_{T_A}, z_B)$ *is* this probability. Intuitively, if there are transitions in the dataset which indicate that finishing (B) from s_{T_A} by taking action a_{T_A} is possible, then this Q-value should be non-zero and the agent will learn to chain (A) and (B) together through their aggregate instruction $\hat{z}$. Our reward labels for the two trajectories with aggregate instruction $\hat{z}$ are therefore:

$$R(s, a, \hat{z}) = \begin{cases} 1, & \text{for } s = s_{T_B} \\ Q(s, a, z_B), & \text{for } s = s_{T_A} \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Since Q changes during training, we compute the rewards in Eq. 3.3 in each batch while training. Full SPRINT pseudocode is listed in Alg. 1.

3.4 Experiments

In our experiments, we investigate how well an agent pre-trained with SPRINT performs on challenging unseen tasks. Thus, we answer the following questions: (1) Does SPRINT enable more efficient finetuning on unseen target tasks than previous pre-training approaches? (2) Can SPRINT agents execute unseen

Algorithm 1 SPRINT Algorithm

Require: Dataset $\mathcal{D}^L$ w/ language instruction labels, LLM
AggregateSkills($\mathcal{D}^L$, LLM)
while not converged **do**
  $\tau_z \leftarrow \mathcal{D}^L$: Sample an annotated skill (sub-)trajectory
  Train offline RL on τ_z
  $\tau_{\text{agg}_1}, \tau_{\text{agg}_2} \leftarrow$ ChainSkills($\mathcal{D}^L$, LLM)
  Train offline RL on $\tau_{\text{agg}_1}, \tau_{\text{agg}_2}$
procedure AggregateSkills($\mathcal{D}^L$, LLM) ▷ Sec. 3.3.2
  for composite trajectory $\tau_{\tilde{z}}$ in $\mathcal{D}^L$ **do**
    for all adjacent sub-trajectories $\left[\tau_{z_i} ... \tau_{z_j}\right]$ **do**
      Assign name from LLM: LLM$(z_i ... z_j) = \hat{z}_{i:j}$
      $\tau_{\hat{z}_{i:j}} \leftarrow$ Concat $\left[\tau_{z_i}, ..., \tau_{z_j}\right]$ and relabel with $\hat{z}_{i:j}$ and reward from Eq. 3.1.
      $\mathcal{D}^L = \mathcal{D}^L \cup \left\{\tau_{\hat{z}_{i:j}}\right\}$
procedure ChainSkills($\mathcal{D}^L$, LLM) ▷ Sec. 3.3.3
  Sample random $\tau_{z_1}, \tau_{z_2} \sim \mathcal{D}^L$
  Assign new name : $\hat{z} =$ "$\{z_1\}$ and $\{z_2\}$"
  $\tau_{\text{agg}_1} \leftarrow$ Relabel τ_{z_1} w/ $\hat{z}$ and rew from Eq. 3.3
  $\tau_{\text{agg}_2} \leftarrow$ Relabel τ_{z_2} w/ $\hat{z}$ and rew from Eq. 3.3
  return $\tau_{\text{agg}_1}, \tau_{\text{agg}_2}$

language instructions zero-shot? (3) Does augmentation via *language* relabeling lead to more generalizable policies than through goal image relabeling?

3.4.1 Experimental Setup

We evaluate our approach on two image-based environments (see Figure 3.4): ALFRED-RL, a simulated RL benchmark we introduce, and a real robot kitchen.

ALFRED-RL. Our goal is to compare different pre-training approaches on a diverse set of semantically meaningful, long-horizon tasks. Yet, existing multi-task RL environments typically evaluate only on short-horizon or semantically meaningless tasks [392, 237]. Thus, we introduce a new RL benchmark based on the ALFRED household task simulator [328]. While ALFRED abstracts away low-level agent control into discrete actions like "pick up" or "turn left," its 100+ rich indoor scenes with many interactable objects allow to evaluate an agent's capabilities for solving long-horizon household tasks from a rich task distribution. The original benchmark focuses on imitation learning, but we extend it to support training RL agents through a gym interface with egocentric RGB observations and an action space consisting of

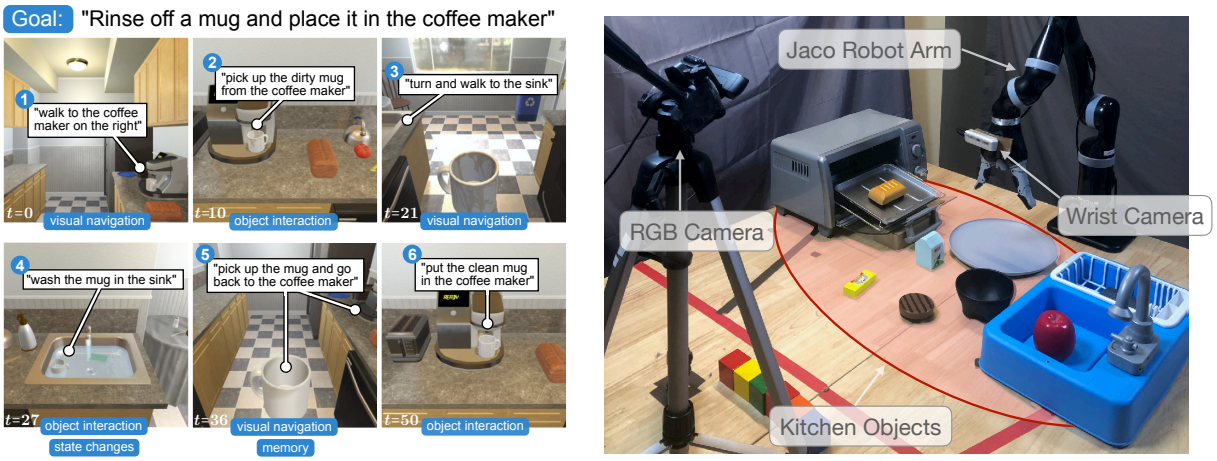


Figure 3.4: **Left**: ALFRED provides a rich set of long-horizon, meaningful tasks and a dataset of 6.6k language-annotated demos. We introduce the ALFRED-RL Benchmark which tests finetuning of RL agents on unseen tasks and scenes. **Right**: Our Jaco robot arm with RGB image-based control.

12 discrete action choices and 82 interactable object types [273]. We create three evaluation task sets that test progressively more challenging axes of generalization: $\textbf{\textit{EVAL}}_{\textbf{\textit{INSTRUCT}}}$ uses unseen human-generated instructions on familiar scenes, $\textbf{\textit{EVAL}}_{\textbf{\textit{LENGTH}}}$ uses tasks that are longer than any observed in pre-training, testing "stitching" capabilities, and $\textbf{\textit{EVAL}}_{\textbf{\textit{SCENE}}}$ uses tasks in unseen floorplans.

Real-World Robot Kitchen Manipulation. To evaluate pre-training approaches on end-to-end *low-level* robot control, we design a set of stylized kitchen manipulation tasks with a Kinova Jaco 2 robot arm. The policy's inputs are RGB images from a wrist-mounted and a third-person camera and it produces continuous end-effector (3-dim) displacement actions and a discrete gripper open/stay/close action at a control frequency of 10Hz. We collect a dataset of 329 long-horizon trajectories via human teleoperation with the setup from Dass et al. [76], each consisting of multiple language-annotated sub-trajectories like *"pick up the apple fruit,"*, *"place the black bowl in the dish rack,"* etc. For evaluation, we construct three long-horizon tasks, sequencing 2 to 8 "primitive skills" like the ones mentioned above, in environment configurations that are unseen in the pre-training data. We collect 25 demonstrations for each of the three tasks to evaluate offline fine-tuning performance of different pre-trained policies.

Comparisons. We compare SPRINT against common policy pre-training approaches, behavioral cloning and offline goal-conditioned RL: **Language-conditioned BC (L-BC)** [150, 218]: Behavior cloning (BC) conditioned on the individual language instructions; **Episodic Transformers (ET)** [273]: BC conditioned on sequences of language instructions – ET is the best-performing end-to-end learned policy on the ALFRED leaderboard that *does not* use privileged domain knowledge like hand-engineered policies or voxel maps; **Actionable Models (AM)** [49]: Goal-conditioned offline RL with randomly sampled goal observations from the same training data as SPRINT. We also evaluate **SayCan** [7]: Top-down LLM planning over pre-trained, language-conditioned policies.

All methods use the same architectures, hyperparameters, and training data $\mathcal{D}^L$ where possible. In ALFRED-RL, all methods use the same language token conditioned transformer policy architecture proposed by Pashevich, Schmid, and Sun [273] specifically for ALFRED; we use a transformer critic model with a separate output head for each critic, following Snell et al. [333]. On the real robot, all methods use an RNN architecture with "action chunking" [411] proposed by Dass et al. [75]. Results are means and standard deviations over 3 seeds.

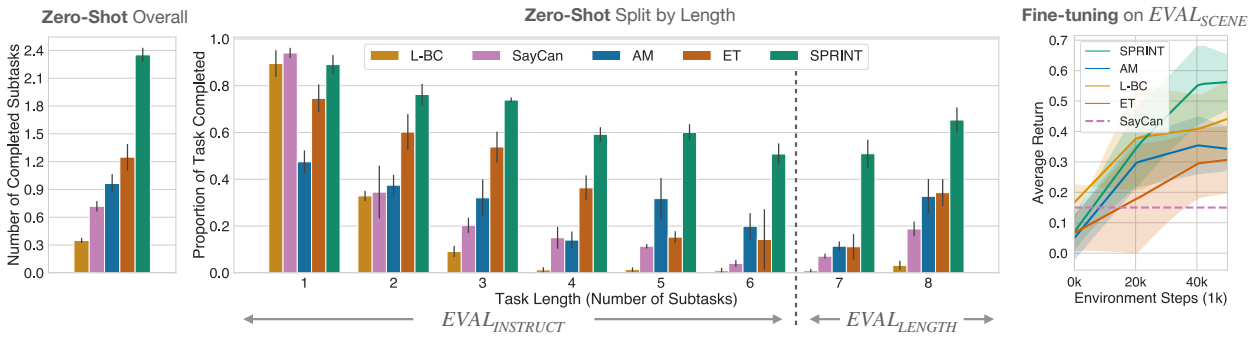


Figure 3.5: ALFRED-RL evaluation results. **Left**: Zero shot performance on $\text{EVAL}_{\text{INSTRUCT}}$ and $\text{EVAL}_{\text{LENGTH}}$. SPRINT is able to complete substantially more subtasks than prior approaches. **Middle**: Breakdown of performance by task length. SPRINT performs well on challenging, long tasks. **Right**: Finetuning performance in unseen floor plans of $\text{EVAL}_{\text{SCENE}}$. SPRINT learns in new floorplans more effectively by reaching higher performance.

3.4.2 SPRINT Solves Long-Horizon Tasks Zero-Shot

We first test the effectiveness of SPRINT's pre-training by analyzing zero-shot performance across 100 unseen tasks in the $EVAL_{INSTRUCT}$ evaluation set. We report results in Figure 3.5 (left). Our approach, SPRINT, achieves **2-8x** higher zero-shot task performance than prior pre-training approaches AM and L-BC. Even though ET also trains to condition on long-horizon instruction sequences like SPRINT, ours still outperforms it overall by 2x. To better understand the differences between the methods, we report the breakdown of returns by length of the evaluation task in Figure 3.5 (middle). We find that all methods except AM achieve similar performance on length 1 tasks. However, on long-horizon tasks, SPRINT achieves much higher returns than all baselines since it can leverage the LLM to automatically generate longer-horizon pre-training tasks. In contrast, L-BC trains only on the human-provided, shorter-horizon annotations and thus cannot zero-shot perform longer tasks. Meanwhile SayCan, with the same LLM as used for SPRINT, commonly generates incorrect plans that lead to incorrect behaviors. This problem is exacerbated on longer tasks; the chance of planning errors increases with task length. In contrast, SPRINT's pre-training enables more robust long-horizon task execution. Similar to our approach, AM trains to reach long-horizon goals during pre-training but the results in Figure 3.5 (left) show that its pre-training with goal-state conditioning is *less* effective than our language-conditioned pre-training. These results also hold for the $EVAL_{LENGTH}$ task set, which tests generalization to task horizons beyond the ones seen during training. On these most challenging tasks, SPRINT outperforms the best baseline by 2.5x.

3.4.3 SPRINT Finetunes Effectively in Unseen Environments

ALFRED-RL. We test SPRINT's finetuning performance to unseen tasks on the most challenging $EVAL_{SCENE}$ task set in unseen household floor plans with 50k environment interactions. This corresponds to a realistic scenario in which an agent is placed in a new household environment and needs to leverage skills learned during pre-training to solve new tasks with minimal environment interaction. To implement finetuning

for SPRINT and AM, we condition the policy on a language instruction or goal image from the target task respectively and then run IQL with online data collection. For L-BC and ET, we first pre-train a language-conditioned critic with IQL on the pre-training dataset and then finetune both the policy and critic with online IQL. Sparse, per-subtask completion reward is given to agents during fine-tuning.

We report finetuning results in Figure 3.5 (right). SPRINT quickly achieves higher downstream task return than the best prior work. Specifically, L-BC converges to lower peak performance than SPRINT and ET performs poorly, perhaps because transferring from instruction sequences to high-level task descriptions is challenging. Meanwhile, AM performs similarly to L-BC, possibly because unseen goal states are more difficult to learn from. In contrast, SPRINT's pre-training with language conditioning allows for effective transfer even to unseen environments since the semantics of the tasks transfer well: the language description "*place cup in coffee machine*" transfers to many environments while the goal image for the same task might look very different. Thus, pre-training with language instructions can enable better transfer for learning tasks in new environments than pre-training to reach goal states. SayCan performs poorly due to both planning and execution errors as it does not fine-tune. We also attempted to first fine-tune SayCan's primitive policies before running SayCan, but its performance did not change as fine-tuning its policies on high-level task instructions did not improve primitive instruction execution.

Real Robot. We also measure finetuning performance on an unseen environment on our real robot setup. We evaluate on three tasks consisting of 2, 4, and 8 subgoals, respectively:

1. *Bake bread in the oven*: The robot must (1) pick up the bread, (2) place it in the oven.
2. *Serve heated milk in the bowl*: The robot must (1) pick up the milk, (2) place it in the black bowl, (3) pick up the bowl with milk, (4) place the bowl in the oven.

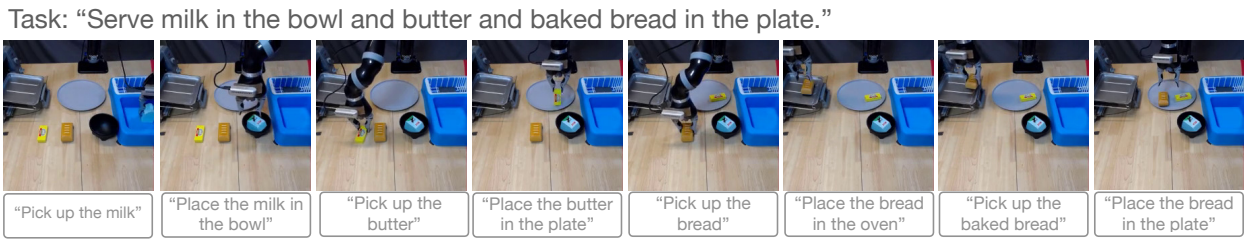


Figure 3.6: Successful rollout of a SPRINT agent offline finetuned for the task above with object combinations not in the pre-training data. SPRINT solves all 8 tasks in sequence.

3. *Serve milk in the bowl and butter and baked bread in the plate*: (1) pick up milk, (2) put it in the black bowl, (3) pick up butter, (4) put it in the plate, (5) pick up the bread, (6) bake it in the oven, (7) pick up the bread from the oven, (8) place the bread in the plate.

We collect 25 demonstrations per task for offline finetuning. We compare SPRINT against **L-BC**, a version of L-BC trained on full sequences of concatenated language instructions (**L-BC Composite**), and a method that is trained only on the downstream task demonstrations (**No pre-train**).

Results in Table 3.1 demonstrate that *No Pre-train* performs poorly, indicating that pre-training is necessary. SPRINT achieves the best success rates and completes the most subgoals on all tasks. Compared to L-BC Composite, SPRINT achieves higher returns and success rates on challenging, longer tasks. See Figure 3.6 for an example evaluation.

Table 3.1: Success rates and number of subgoals completed after fine-tuning on the tabletop arrangement displayed on the left with unseen object combinations over 5 trials.

	Length 2		Length 4		Length 8	
Method	Success	# Tasks	Success	# Tasks	Success	# Tasks
SPRINT	**100%**	**2.0**	**60%**	**3.4**	**40%**	**6.2**
L-BC Comp.	**100%**	**2.0**	40%	2.8	20%	5.2
L-BC	**100%**	**2.0**	40%	0.4	0%	2.0
No pre-train	0%	1.0	0%	0.0	0%	0.0

3.4.4 Ablation Studies

We verify the effectiveness of the components of our approach, with the following ablations: **SPRINT w/o chain** removes cross-trajectory chaining (Section 3.3.3), instead trains only on within-trajectory human-provided and LLM-aggregated tasks; **SPRINT Naïve Chain** replaces Q-value reward labels when chaining

with 0's to test naïve offline RL "stitching" with language instruction-conditioned agents. **SPRINT w/o LLM-agg** additionally removes LLM aggregation (Section 3.3.2) and chaining, thus training only on the human-provided task annotations. We report zero-shot ALFRED evaluation results in Table 3.2: each component of our approach improves zero-shot evaluation performance. There is a large performance loss when removing LLM aggregation, underlining the importance of leveraging LLMs for automatically generating long-horizon training tasks. We also see that naïve chaining is worse than not chaining.

3.5 Discussion and Acknowledgements

We presented SPRINT, an approach for scalable agent pre-training that automatically generates training tasks for offline RL via LLM relabeling and cross-trajectory skill chaining. SPRINT pre-training leads to higher zero-shot and finetuning performance on diverse household tasks in the ALFRED simulator and on real-robot kitchen manipulation tasks.

Table 3.2: Ablations. SPRINT achieves the highest return.

Ablation	$EVAL_{INSTRUCT}$	$EVAL_{LENGTH}$
SPRINT (ours)	**1.94 ± 0.04**	**4.40 ± 0.39**
SPRINT w/o Chain	1.75 ± 0.11	3.98 ± 0.29
SPRINT Naïve Chain	0.50 ± 0.04	0.26 ± 0.05
SPRINT w/o LLM-agg	0.37 ± 0.01	0.15 ± 0.10

Chapter 4

EXTRACT: Efficient Policy Learning by Extracting Transferable Robot Skills from Offline Data

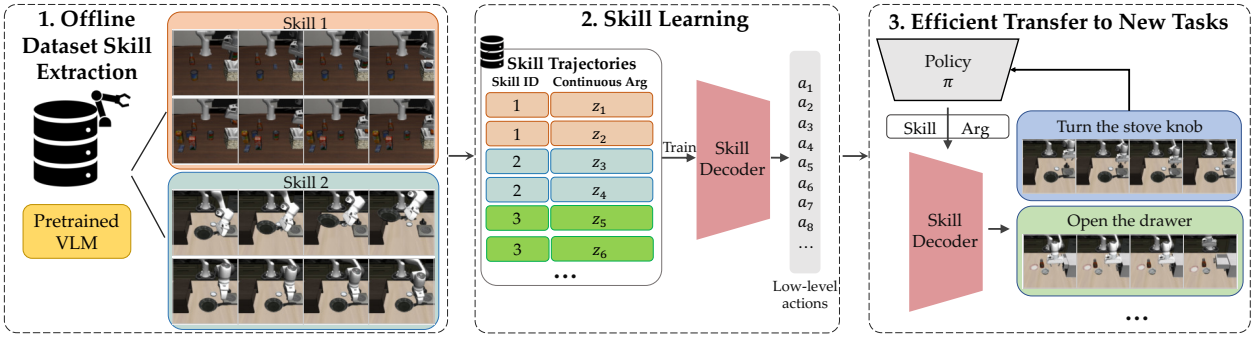


Figure 4.1: EXTRACT unsupervisedly extracts a discrete set of skills from offline data that can be used for efficient learning of new tasks. (1) EXTRACT first uses VLMs to extract a discrete set of aligned skills from image-action data. (2) EXTRACT then trains a skill decoder to output low-level actions given discrete skill IDs and learned continuous arguments. (3) This decoder helps a skill-based policy efficiently learn new tasks with a simplified action space over skill IDs and arguments.

4.1 Introduction

Imagine learning to play racquetball as a complete novice. Without prior experience in racket sports, this poses a daunting task that requires learning not only the (1) complex, high-level strategies to control *when* to serve, smash, and return the ball but also (2) *how* to actualize these moves in terms of fine-grained motor control. However, a squash player should have a considerably easier time adjusting to racquetball as they already know how to serve, take shots, and return; they simply need to learn *when* to use these skills and

how to adjust them for larger racquetball balls. Our paper aims to make use of this intuition to enable efficient learning of new robotics tasks.

In general, humans can learn new tasks quickly—given prior experience—by adjusting existing skills for the new task [102, 12]. Skill-based reinforcement learning (RL) aims to emulate this transfer [343, 306, 131, 282, 403, 8, 72, 260, 399, 405] in learned agents by equipping them with a wide range of skills (i.e., temporally-extended action sequences) that they can call upon for efficient downstream learning. Transferring to new tasks in standard RL, based on low-level environment actions, is challenging because the learned policy becomes more task-specific as it learns to solve its training tasks [309, 349, 344, 95, 46]. In contrast, skill-based RL leverages temporally extended skills that can be both transferred across tasks and yield more informed exploration [282, 329, 406], thereby leading to more effective transfer and learning. However, existing skill-based RL approaches rely on costly human supervision [183, 325, 72, 260] or restrictive skill definitions [118, 282, 8] that limit the expressiveness and adaptability of the skills. Therefore, we ask: how can robots discover *adaptable* skills for efficient transfer learning *without costly human supervision*?

Calling back to the squash to racquetball transfer example, we humans categorize different racket movements into *discrete skills*—for example, a "forehand swing" is distinct from a "backhand return." These discrete skills can be directly transferred by making minor modifications for racquetball's larger balls and different rackets. This process is akin to that of calling a programmatic API, e.g., `def forehand(x, y)`, where learning to transfer reduces to learning *when* to call discrete functions (e.g., `forehand()` vs `backhand()`) and *how* to execute them (i.e., what their arguments should be). In this paper, we propose a method to accelerate transfer learning by enabling robots to learn, without expert supervision, a discrete set of skills parameterized by input arguments that are useful for downstream tasks (see Figure 4.1). We assume access to an offline dataset of image-action pairs of trajectories from tasks that are different from the downstream target tasks. Our key insight is aligning skills by extracting *high-level behaviors*, i.e., discrete skills like

"forehand swing," from images in the dataset. However, two challenges preclude realizing this insight: (1) how to extract these input-parameterized skills, and (2) how to guide online learning of new tasks with these skills.

To this end, we propose EXTRACT (**Ex**traction of **T**ransferable **R**obot **Act**ion Skills), a framework for extracting discrete, parameterized skills from offline data to guide online learning of new tasks. We first use pre-trained vision-language models (VLMs), trained to align images with language descriptions [289] so that images of similar high-level behaviors are embedded to similar latent embeddings [334], to extract—from our offline data—image embedding differences representing changes in high-level behaviors. Next, we cluster the embeddings in an *unsupervised* manner to form discrete skill clusters that represent high-level skills. To parameterize these skills, we train a *skill decoder* on these clusters, conditioned on the skill ID (e.g., representing a "backhand return") and a learned argument (e.g., indicating velocity), to produce a skill consisting of a temporally extended, variable-length action sequence. Finally, to train a robot for new tasks, we train a skill-based RL policy to act over this skill-space while being guided by skill prior networks, learned from our offline skill data, guiding the policy for (1) *when* to select skills and (2) *what* their arguments should be.

In summary, EXTRACT enables sample-efficient transfer learning for robotic tasks by extracting a meaningful set of skills from offline data for an agent to use for learning new tasks. We first validate that EXTRACT learns a well-clustered set of skills. We then perform experiments across challenging, long-horizon, sparse-reward, image-based robotic manipulation tasks, both in simulation and in the real world on a Panda Franka arm, demonstrating that EXTRACT agents can more quickly transfer skills to new tasks than prior work.

4.2 Related Work

Defining Skills Manually. Many works require manual definition of skills, e.g., as pre-defined primitives [306, 274, 196], subskill policies [268, 183, 379], or task sketches [15, 325], making them challenging to scale to arbitrary environments. Closest to ours, Dalal, Pathak, and Salakhutdinov [72] and Nasiriany, Liu, and Zhu [260] hand-define a set of skills parameterized by continuous arguments. But this hand-definition requires expensive human supervision and task-specific, environment-specific, or robot-specific fine-tuning. In contrast, EXTRACT *automatically* learns skills from offline data, which is much more scalable to enable learning multiple downstream tasks. We demonstrate in Section 4.5 that, given sufficient data coverage, skills extracted from data can transfer as effectively as hand-defined skills.

Unsupervised Skill Learning. A large body of prior work discovers skills in an unsupervised manner to accelerate learning new tasks. Some approaches use heuristics to extract skills from offline data, like defining skills as randomly sampled trajectories [168, 318, 241, 317, 217, 282, 8, 403, 324]. While these approaches have demonstrated that randomly sampled skill sequences can accelerate downstream learning, EXTRACT instead uses visual embeddings from VLMs to combine sequences performing similar behaviors into the same skill while allowing for intra-skill variation through their arguments. We show in Section 4.5 that our skill parameterization allows for more efficient online learning than randomly assigned skills. Moreover, Wan et al. [363] also learns skills via clustering visual features; however, in addition to major differences in methodology, they focus on *imitation* learning—requiring significant algorithmic changes to facilitate learning new tasks online [251, 175, 413]. Instead, we directly focus on online *reinforcement* learning of new tasks.

Another line of work aims to discover skills for tasks without offline data. Some learn skills while simultaneously attempting to solve the task [343, 19, 131, 249, 406, 399]. However, learning the skills and using them simultaneously is challenging, especially without dense reward supervision. Finally, some prior works construct unsupervised objectives, typically based on entropy maximization, to learn task-agnostic

behaviors [93, 370, 113, 320, 178]. However, these entropy maximization objectives lead to learning a large set of skills, most of which form random behaviors unsuitable for any meaningful downstream task. Thus, using them to learn long-horizon, sparse-reward tasks is difficult. We focus on first extracting skills from demonstration data, assumed to have meaningful behaviors to learn from, for online learning of unseen, sparse-reward tasks.

4.3 Preliminaries

Problem Formulation. We assume access to an offline dataset of trajectories $\mathcal{D} = \{\tau_1, \tau_2, ...\}$ where each trajectory consists of ordered image observation and action tuples, $\tau_i = [(s_1, a_1), (s_2, a_2), ...]$. The downstream transfer learning problem is formulated as a Markov Decision Process in which we want to learn a policy π to maximize downstream rewards. We note that the offline dataset $\mathcal{D}$ does not contain trajectories from downstream task(s); we assume that the state space $\mathcal{S}$ has the same dimensions and that actions in $\mathcal{D}$ can be used to solve downstream tasks.

SPiRL. In order to extract skills from offline data and use these skills for a new policy, we build on top of a previous skill-based RL method, namely SPiRL [282]. SPiRL focused on learning skills defined by randomly sampled, fixed-length action sequences. We briefly summarize SPiRL here: Given H-length sequences of consecutive actions from $\mathcal{D}$: $\bar{a} = a_1, ..., a_H$, SPiRL learns (1) a generative **skill decoder** model, $p_a(\bar{a} \mid z)$, which decodes learned, latent skills z encoded by a **skill encoder** $q(z \mid \bar{a})$ into environment action sequences $\bar{a}$, and (2) a state-conditioned skill **prior** $p_z(z \mid s)$ that predicts which latent skills z are likely to be useful at state s. To learn a new task, SPiRL trains a skill-based policy $\pi(z \mid s)$, whose outputs z are skills decoded by $p_a(\bar{a} \mid z)$ into low-level environment actions. The objective of policy learning is to maximize returns under $\pi(z \mid s)$ with a KL divergence constraint to regularize π against the prior $p_z(z \mid s)$.

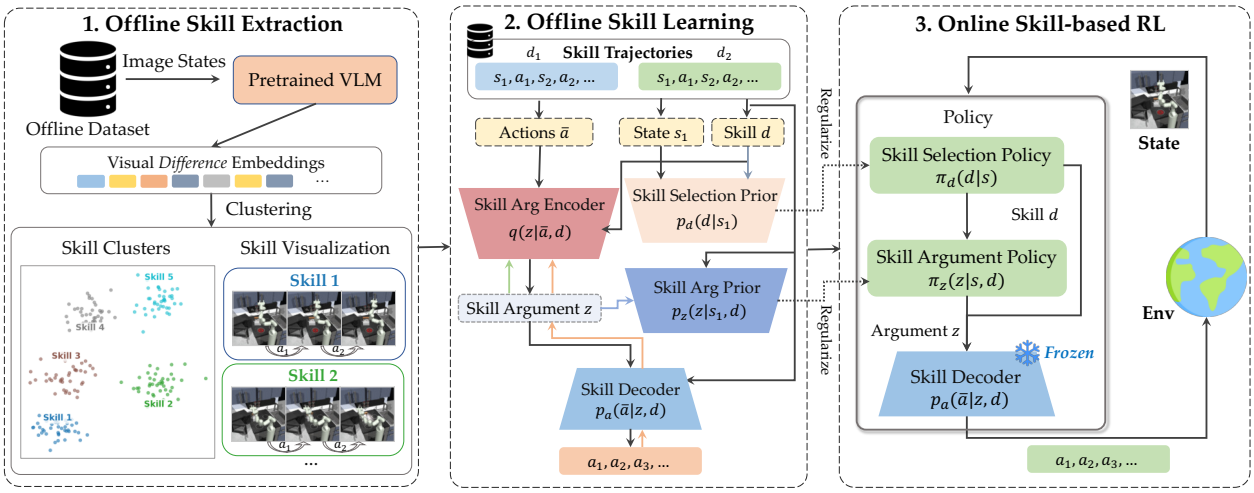


Figure 4.2: EXTRACT consists of three phases. **(1) Skill Extraction**: We extract a discrete set of skills from offline data by clustering together visual VLM difference embeddings representing high-level behaviors. **(2) Skill Learning**: We train a skill decoder model, $p_a(\bar{a} \mid z, d)$, to output variable-length action sequences conditioned on a skill ID d and a learned continuous argument z. The argument z is learned by training $p_a(\bar{a} \mid z, d)$ with a VAE reconstruction objective from action sequences encoded by a skill encoder, $q(z \mid \bar{a}, d)$. We additionally train a skill selection prior and skill argument prior $p_d(d \mid s)$, $p_z(z \mid s, d)$ to predict which skills d and their arguments z are useful for a given state s. Colorful arrows indicate gradients from reconstruction, argument prior, selection prior, and VAE losses. **(3) Online RL**: To learn a new task, we train a skill selection and skill argument policy with RL while regularizing them with the skill selection and skill argument priors.

4.4 Method

EXTRACT aims to discover a discrete skill library from an offline dataset that can be modulated through input arguments for learning new tasks efficiently. EXTRACT operates in three stages: (1) an offline skill *extraction* stage, (2) an offline skill *learning* phase in which we train a decoder model to reproduce action sequences given a skill choice and its arguments, and finally (3) the online RL stage for training an agent to utilize these skills for new tasks. See Figure 4.2 for a detailed overview.

4.4.1 Offline Skill Extraction

Feature extraction. We leverage vision-language models (VLMs), trained to align large corpora of images with natural language descriptions [289, 252, 375, 221], to extract high-level features used to label skills. Although our approach does not require the use of language, we utilize VLMs because, as VLMs were

trained to align images with *language*, VLM image embeddings represent a *semantically aligned* embedding space. However, one main issue precludes the naïve application of VLMs in robotics. In particular, VLMs do not inherently account for object variations or robot arm starting positions across images [70, 334, 299, 367]. But in robot manipulation, high-level behaviors should be characterized by *changes* in arm and object positions across a trajectory—picking up a cup should be considered the same skill regardless of if the cup is to the robot's left or right. Our initial experiments of using the embeddings directly resulted in skills specific to one type of environment layout or object. Therefore, to capture high-level behaviors, we use trajectory-level *embedding differences* by taking the difference of each VLM image embedding with the first one in the trajectory:*

$$e_t = \text{VLM}(s_t) - \text{VLM}(s_1). \tag{4.1}$$

Skill label assignment. After creating embeddings e_t for each image s_t, we assign skill labels in an *unsupervised* manner based on these features. Inspired by classical algorithms from *speaker diarization*, a long-studied problem in speech processing where the objective is to assign a "speaker label" to each speech timestep [16], we first perform unsupervised clustering with K-means on the entire dataset of embedding differences e_i to assign per-timestep skill labels (the label is the cluster ID), then we smooth out the label assignments with a simple median filter run along the trajectory sequence to reduce the frequency of single or few-timestep label assignments. See Figure 4.3 for a visual demonstration of this process.

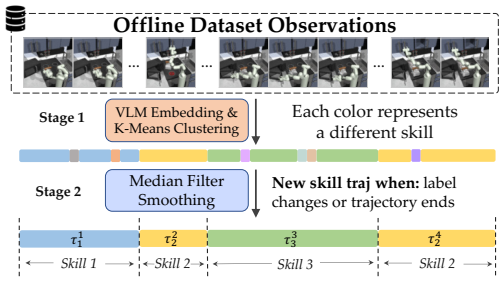


Figure 4.3: Skill label assignment consists of (1) using the VLM embedding differences for clustering, then (2) applying a median filter over the labels to smooth out noisy assignments.

*To ensure that each timestep has an embedding, we assign embedding e_1 to be identical to e_2.

In summary, we first extract observation embedding difference features with a VLM and then perform unsupervised K-means clustering to obtain skill labels for each trajectory timestep. This forms the skill-labeled dataset $\mathcal{D}_d = \{\tau_d^1, \tau_d^2, ...\}$, where each τ_d is a trajectory of sequential (s, a) tuples that all belong to one skill d. Next, we perform skill learning on $\mathcal{D}_d$.

4.4.2 Offline Skill Learning

We aim to learn a *discrete* set of skills, parameterized by *continuous* arguments, similar to a functional API over skills (see Figure 4.2 middle). Therefore, we train a generative **skill decoder** $p_a(\bar{a} \mid z, d)$ to convert a discrete skill choice d and a continuous argument for that skill, z, into an action sequence. As alluded to in Section 4.3, we build upon SPiRL by Pertsch, Lee, and Lim [282]. However, they train their decoder to decode fixed-length action sequences from a single continuous latent z. In contrast, we automatically extract a set of variable-length skill trajectories with labels denoted d and parameterize each skill by a learned, continuous latent argument z.[†]

We train an autoregressive VAE [167] consisting of the following learned neural network components: a skill argument **encoder** $q(z \mid \bar{a}, d)$ mapping to a continuous latent z conditioned on a discrete skill choice d and an action sequence $\bar{a}$, and an autoregressive skill **decoder** $p_a(\bar{a} \mid z, d)$ conditioned on the latent z and the discrete skill choice d.[‡] Because the action sequence $\bar{a}$ can be of various lengths, the decoder also learns to produce a continuous value l at each autoregressive timestep representing the proportion of the skill completed at the current action. This variable is used during online RL to stop the execution of the skill when l equals 1 (see Appendix B.2.1 for further details).

Recall that SPiRL also trains a skill prior network $p_z(z \mid s)$ that predicts which z is useful for an observation s; this prior is used to guide a high-level policy toward selecting reasonable z while performing RL. In contrast with SPiRL where z uniquely represents a skill, we train *two* prior networks, one to guide

[†]To simplify notation, we use z for both our method and SPiRL. However, it is important to note that z uniquely determines the skill in SPiRL, while z denotes a continuous latent argument in our method.

[‡]$p_a(\bar{a} \mid z, d)$ can also be state-conditioned. We opt not to for better *transfer* to new tasks with unseen states.

the selection of the skill d, $p_d(d \mid s)$, and one to guide the selection of its argument z given d, $p_z(z \mid s, d)$. These are trained with the observation from the first timestep of the sampled trajectory, s_1, to be able to guide a skill-based policy during online RL in choosing d and z. Our full objective for training this VAE is to maximize the following:

$$\mathop{\mathbb{E}}_{\substack{\bar{a},d,s_1 \sim \mathcal{D}_d \\ z \sim q(\cdot \mid \bar{a}, d)}} \Bigg[\Bigg[\sum_{t=1}^{|\bar{a}|} \underbrace{\log p_a(a_t, l \mid z, d)}_{\text{action rec. + progress pred.}} \Bigg] + \underbrace{\beta \, \mathrm{KL}\left(q(z \mid \bar{a}, d) \parallel N(0, I)\right)}_{\text{VAE encoder KL regularization}} + \underbrace{\log p_d(d \mid s_1)}_{\text{discrete skill prior}} + \underbrace{\log p_z(\mathbf{sg}(z) \mid s_1, d)}_{\text{continuous arg. prior}} \Bigg], \tag{4.2}$$

where the stop-gradient $\mathbf{sg}(\cdot)$ prevents prior losses from influencing the encoder and z is sampled from the encoder $q(z \mid \bar{a}, d)$. The first two terms are the β-VAE objective [137]; the last two train priors to predict the correct skill d and continuous argument z given s_1.

Additional fine-tuning. On extremely challenging transfer scenarios, demonstrations may still be needed to warm-start reinforcement learning [355]. EXTRACT can also flexibly be applied to this setting by using the same K-means clustering model from Section 4.4.1, which was trained to cluster $\mathcal{D}_d$, to assign skill labels to an additional, smaller demonstration dataset. After pre-training on $\mathcal{D}_d$, we then fine-tune the entire model on that labeled demonstration dataset before performing RL.

4.4.3 Online Skill-Based Reinforcement Learning

Finally, we describe how we perform RL for new tasks by training a skill-based policy to select skills and their arguments to solve new tasks. See Figure 4.2, right, for an overview of online RL.

Policy parameterization. After pre-training the decoder $p_a(\bar{a} \mid z, d)$, we treat it as a frozen lower-level policy that a learned skill-based policy can use to interact with a new task. Specifically, we train a skill-based policy $\pi(d, z \mid s)$ to output a (d, z) tuple representing a discrete skill choice and its continuous argument. We parameterize this policy as a product of two policies: $\pi(d, z \mid s) = \pi_d(d \mid s)\pi_z(z \mid s, d)$ so that each component of $\pi(d, z \mid s)$ can be regularized with our pre-trained priors $p_d(d \mid s)$ and $p_z(z \mid s, d)$.

Intuitively, this parameterization separates decision-making into *what* skill to use and *how* to use it. The complete factorization of the skill-based policy follows:

$$\pi(a \mid s) = \underbrace{p_a(\bar{a} \mid z, d)}_{\text{skill decoder}} \cdot \pi(d, z \mid s) = \underbrace{p_a(\bar{a} \mid z, d)}_{\text{skill decoder}} \cdot \underbrace{\pi_d(d \mid s) \cdot \pi_z(z \mid s, d)}_{\text{learned skill-based policy}}. \tag{4.3}$$

Policy learning. We can train the skill-based policy with online data collection using any entropy-regularized RL algorithm, such as SAC [123] or RLPD [23], where we regularize against the skill priors instead of against a max-entropy uniform prior. Because we have factorized $\pi(d, z \mid s)$ into two separate policies, we can easily regularize each with the priors trained in Section 4.4.2. The training objective for the policy with SAC is to maximize over π_d, π_z:

$$\mathop{\mathbb{E}}_{\substack{s,d\sim\pi_d(.|s)\\ z\sim\pi_z(.|s,d)}}\Big[Q(s,z,d) - \alpha_z \underbrace{\mathrm{KL}(\pi_z(z \mid s,d) \parallel p_z(\cdot \mid s,d))}_{\text{skill argument guidance}} - \alpha_d \underbrace{\mathrm{KL}(\pi_d(d \mid s) \parallel p_d(\cdot \mid s))}_{\text{skill choice guidance}}\Big], \tag{4.4}$$

where α_z and α_d control the prior regularization weights. The critic objective is also correspondingly modified (see Appendix Algorithm 6). Despite the hierarchical architecture, this objective is stable to train as the lower-level skill decoder is frozen and the priors regularize the high-level policy.

In summary, EXTRACT first extracts a set of discrete skills from offline image-action data (Section 4.4.1), then trains an action decoder to take low-level actions in the environment conditioned on a discrete skill and continuous latent (Section 4.4.2), and finally performs prior-guided reinforcement learning over these skills online in the target environment to learn new tasks (Section 4.4.3). See Algorithm 3 (appendix) for the pseudocode and Appendix B.2.1 for additional implementation details.

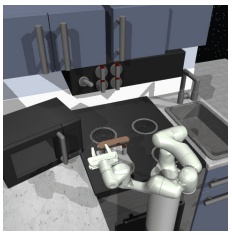
4.5 Experiments

Our experiments investigate the following questions: (1) Does EXTRACT discover meaningful, well-aligned skills from offline data? (2) Do EXTRACT-acquired skills help robots learn new tasks? (3) What components of EXTRACT are important in enabling transfer?

4.5.1 Experimental Setup

We evaluate EXTRACT on two long-horizon, continuous-control, robotic manipulation domains: Franka Kitchen [104] and LIBERO [201]. and the real-world FurnitureBench [135]. All environments use image observations and sparse rewards. For both Franka Kitchen and LIBERO, our method **EXTRACT** uses the R3M VLM [252] and K-means with $K = 8$ for offline skill extraction (Section 4.4.1). In FurnitureBench, $K = 6$. We list specific details below; see Appendix B.2.3 for more.

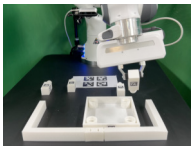
Franka Kitchen: This environment, originally from Gupta et al. [118] and Fu et al. [104] contains a Franka Panda arm operating in a kitchen environment. Similarly to Pertsch, Lee, and Lim [282], we test transfer learning of a sequence of 4 subtasks never performed in sequence in the dataset. Agents are given a reward of 1 for completing each subtask.



LIBERO: LIBERO [201] consists of a Franka Panda arm interacting with many objects and drawers. We test transfer to four task suites, LIBERO-`{Object, Spatial, Goal, 10}` consisting of 10 unseen environments/tasks each, spanning various transfer scenarios (40 total tasks). LIBERO tasks are language conditioned (e.g., "turn on the stove and put the moka pot on it"); for pre-training and RL, we condition all methods on the language instruction. Due to LIBERO's difficulty [210], for all pre-trained methods, we first fine-tune to a

provided additional target task dataset with 50 demos per task before performing RL. During RL, we fine-tune on all tasks within each suite simultaneously. To the best of our knowledge, we are the first to report successful RL results on LIBERO tasks.

FurnitureBench: FurnitureBench [135] tests an agent's ability to assemble real-world furniture with a Franka Panda arm. We pre-train on `one-leg` assembly data without initial object placement randomness and test real-world RL transfer to the same task with ± 5cm of initial object and end-effector position randomness, plus ± 15 degrees of end-effector angle randomization. We use RLPD [23], a sample-efficient actor-critic RL algorithm, for more efficient real-world training. RL is run for 100 training trajectories after pre-training on 500 demonstration trajectories.



Baselines and Comparisons. We compare: (1) an oracle (**RAPS** [72]), which is given *ground truth* discrete skills, with continuous input arguments, designed by humans specifically for Franka Kitchen; (2) methods that pre-train with the same data—namely **SPiRL** [282] which extracts sequences of fixed-length random action trajectories as skills, EXTRACT-**UVD** which replaces our discrete skill extraction with UVD's VLM-based mechanism [409], and **BC**, behavior cloning using the same offline data but no temporally extended skills; and (3) **SAC** [123], i.e., RL without any offline data. See Appendix B.2 for implementation details. Sim results include standard deviations over 5 seeds.

4.5.2 Offline Skill Extraction

We first test EXTRACT's ability to discover meaningful, well-aligned skills during skill extraction. In Figure 4.4, we plot K-means ($K = 8$) skill assignments in Franka Kitchen. We project VLM embedding differences down to 2-D with PCA for visualization. These skill assignments demonstrate that unsupervised clustering of VLM embedding differences can create distinctly separable clustering assignments. For example, skill 4 (Figure 4.4, top left) demonstrates a cabinet opening behavior. See additional visualizations for

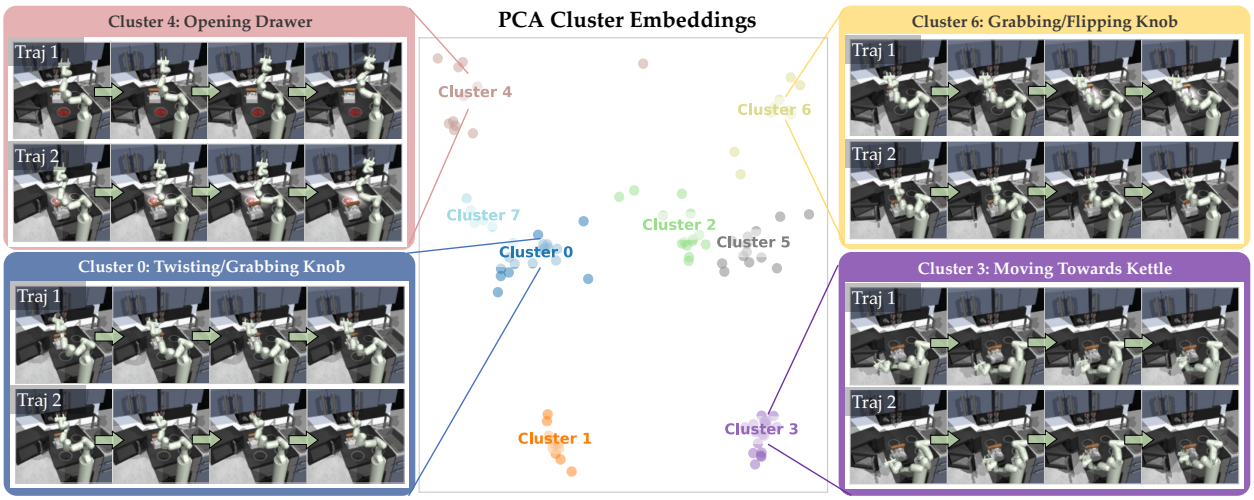


Figure 4.4: 100 randomly sampled trajectories from the Franka Kitchen dataset after being clustered into skills and visualized in 2D (originally 2048) with PCA. Even in 2 dimensions, clusters can be clearly distinguished. We visualize 2 randomly sampled skills in each cluster, demonstrating that our skill assignment mechanism successfully aligns trajectories performing similar high-level behaviors.

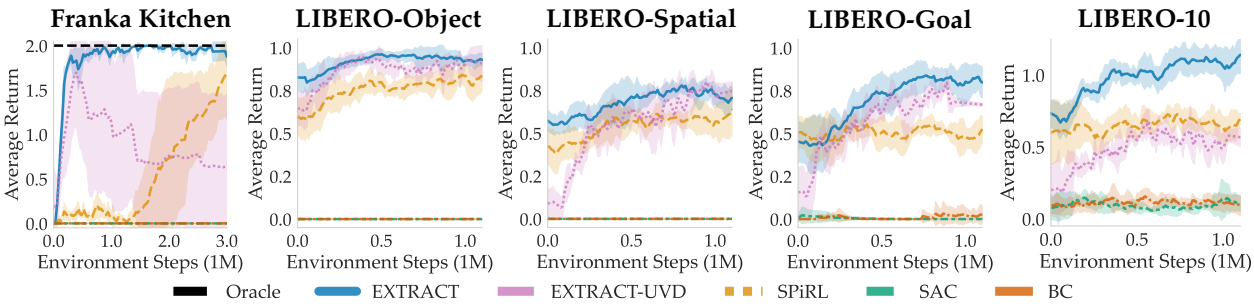


Figure 4.5: SPRINT outperforms SPiRL and EXTRACT-UVD in RL across all comparisons, demonstrating the advantages of our clustered skill-space. SAC and BC struggle, demonstrating the need for skill-based RL. In LIBERO-`{Object, Spatial, Goal}`, return is success rate.

all environments in Appendix B.3.1. We also analyze quantitative clustering statistics in Appendix B.3.2.

Next, let's see how these skills help with learning new tasks.

4.5.3 Online Reinforcement Learning of New Tasks

Simulated Envs. We investigate the ability of all methods to transfer to new tasks in simulation in Figure 4.5. In Kitchen, EXTRACT matches the oracle performance while being **10x** more sample-efficient than SPiRL, with SPiRL needing 3M timesteps to reach EXTRACT's performance at 300k. In all LIBERO suites, EXTRACT performs best in either sample efficiency or final performance due to its discrete-continuous

skill separation enabling easier downstream RL; it outperforms SPiRL and EXTRACT-UVD the most in LIBERO-10, the suite with the longest-horizon tasks. EXTRACT-UVD is unstable in Franka Kitchen (see Appendix B.3.4 for analysis) and generally performs worse as UVD's skill extraction mechanism does not perform our discrete skill clustering. Meanwhile, SAC and BC perform poorly, indicating our tasks are difficult to solve with standard RL or without skills.

Our method outperforms others due to its semantically aligned, discrete skill-space. For example, to open drawers, EXTRACT's policy only needs to learn a single discrete drawer-opening skill when the gripper is near any drawer. In contrast, SPiRL requires memorizing and distinguishing continuous skills for each specific drawer-opening behavior. Additionally, EXTRACT allows easier exploration later in the task, enabling the policy to reuse the same skill for other drawers. For more details, see Appendix B.5. Next, we conduct an ablation study on EXTRACT's components.

Real world. Finally, we assess EXTRACT's real-world performance on FurnitureBench for `one-leg` assembly in Table 4.1. We report the average completed subtask (20 trials) out of a maximum of 5. EXTRACT outperforms SPiRL both before and after 100 episodes of real-world RL fine-tuning, showing effective skill transfer. Overall, EXTRACT excels across 42 tasks and 3 domains, outperforming other skill-based RL, BC, and online RL methods, both in simulation and on robots.

Table 4.1: Furniture RL.

Method	Start	End
SPiRL	1.35	1.55
EXTRACT	**1.90**	**2.50**

4.5.4 EXTRACT RL Ablation Studies

VLMs. We first ablate the use of VLMs from selecting features for clustering. Therefore, we compare against **Action**, where skill labels are generated by clustering robot *action differences*. We also compare against **State** where skills are labeled by clustering ground truth state differences (e.g., robot joints, states of all objects). State represents an oracle scenario as ground truth states of all relevant objects are

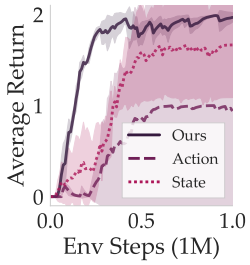


Figure 4.6: Embedding ablations.

difficult to obtain in the real world. We plot results in Franka Kitchen in Figure 4.6. EXTRACT with VLM-extracted skills performs best, as both ground truth state and raw environment action differences can be difficult to directly obtain *high-level*, semantically meaningful skills from. For ablations against pure proprioception and CLIP [289], see Appendix B.3.3.

Number of Clusters. Finally, we ablate the number of K-means clusters. Too few or too many clusters can affect RL by balancing the ease of selecting the correct discrete skill against the complexity of choosing the right continuous argument. In Figure 4.7, we show average returns at 1M timesteps for EXTRACT in Kitchen with $K = 3, 5, 8, 15$. Performance remains stable, with a drop only at $K = 15$, indicating that EXTRACT is robust to variations in the number of discovered discrete skills.

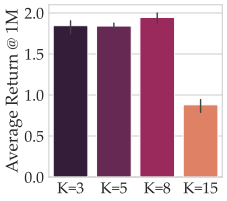


Figure 4.7: Kitchen K ablations.

4.6 Discussion

We presented EXTRACT, a method for enabling efficient agent transfer learning by extracting a discrete set of input-argument parameterized skills from offline data for a robot to use in new tasks. Compared to standard RL, our method operates over temporally extended skills rather than low-level environment actions, providing greater flexibility and transferability to new tasks, as demonstrated by our comprehensive experiments.

Chapter 5

HAMSTER: Hierarchical Action Models for Open-World Robot Manipulation

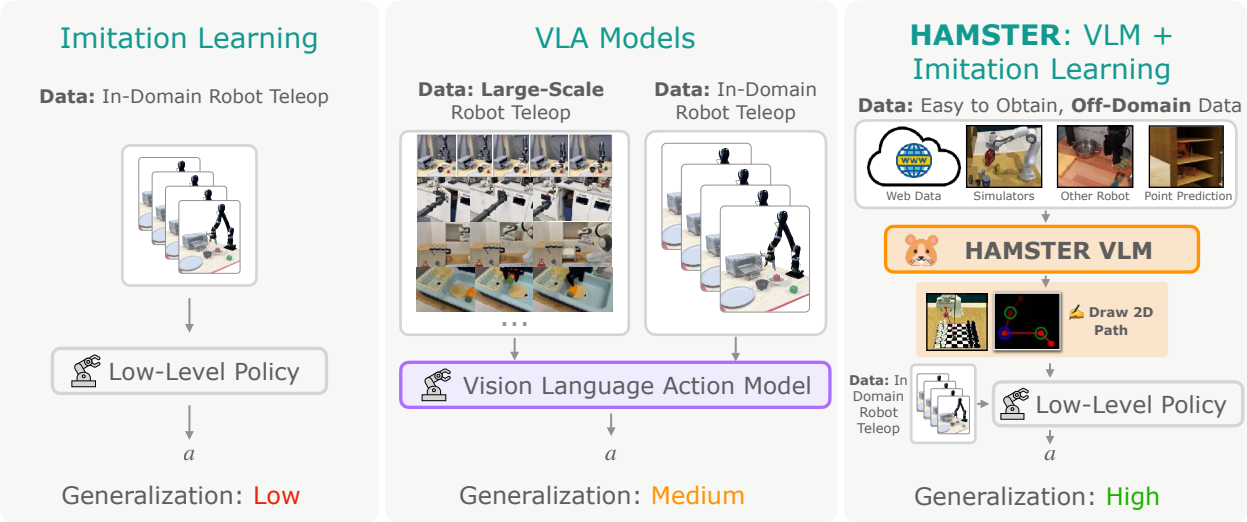


Figure 5.1: Overview of HAMSTER, VLAs and "smaller" imitation learning methods. HAMSTER's hierarchical design results in better generalization with a small amount of in-domain data. HAMSTER is able to utilize cheap training sources such as videos or simulations for enhanced generalization.

5.1 Introduction

Developing general robot manipulation policies has been notoriously difficult. With the advent of large vision-language models (VLMs) that display compelling generalization capabilities, there is optimism that the same recipe is directly applicable to robot manipulation. A line of prior work [36, 165, 32] builds

open-world vision-language-action models (VLAs) by finetuning off-the-shelf pretrained VLMs to directly produce robot actions. These VLA models, which we refer to in this work as *monolithic* VLA models, rely crucially on large robotics datasets, complete with on-robot observations, e.g., images and proprioceptive states, and actions. However, on-robot data is expensive, since end-to-end observation-action pairs are typically collected on the robot hardware through, e.g., teleoperation. Despite recent community-wide efforts in building large-scale robotics datasets [67, 163], the size, quality, and diversity of existing robotics datasets are still limited, and monolithic VLA models have yet to demonstrate emergent capability comparable to VLMs and LLMs in other domains of study. Moreover, monolithic VLA models are constrained by their inference frequency to achieve dexterous and dynamic manipulation tasks [36, 165].

On the other hand, relatively small robot policy models have shown impressive dexterity and robustness. Such models have demonstrated promise across a range of complex tasks involving contact-rich manipulation and 3D reasoning, spanning domains from tabletop manipulation [327, 111, 110, 162] to fine dexterous manipulation [57, 411]. Trained on relatively small datasets, these models show local robustness, and can achieve dexterous and high-precision control. However, they are often brittle to drastic changes in the environment or semantic description of the tasks [288]. These models also can struggle to effectively leverage simulation data for real-world manipulation tasks due to sim-to-real gaps in visual appearances and system dynamics [191, 233].

In this work, we ask – how can we marry the generalization benefits of large VLMs, with the efficiency, local robustness, and dexterity of small policy models? Our key insight is that, instead of directly predicting robot actions, VLMs can be fine-tuned to produce intermediate representations as high-level guidance on solving the robot manipulation task. The intermediate representation can then be consumed by the low-level policy model to produce actions, alleviating the low-level policy from the burden of long-horizon planning and complex, semantic reasoning. Further, if the intermediate representations are chosen such that they are *1)* easily obtainable from image sequences; *2)* largely embodiment agnostic; and *3)* sufficiently

robust to subtle changes in dynamics, the VLM can be fine-tuned with *off-domain* data where robot actions are unavailable or inaccurate. Such off-domain data does not need to be collected on the actual robot hardware. Examples of off-domain data include action-free video data, simulation data, human videos, and videos of robot with different embodiments. These off-domain data are generally easier to collect and may already be abundant in existing datasets. We hypothesize, and show experimentally in Fig 5.7, that this hierarchical separation can allow VLA models to more effectively bridge the domain gap between off-domain data and in-domain robotic manipulation.

To this end, we propose a hierarchical architecture for VLAs, HAMSTER (**H**ierarchical **A**ction **M**odels with **S**epara**TE**d Path **R**epresentations), where large fine-tuned VLMs are connected to low-level policy models via 2D path representations[*]. A 2D path is a coarse trajectory of the 2D image-plane position of the robot end-effector[†], as well as where the gripper state changes, i.e., opens and closes (see Fig. 5.2). These 2D paths can be obtained cheaply and automatically from data sources such as action-free videos or physics simulations, using point tracking [82, 159], hand-sketching [115], or proprioceptive projection. This allows HAMSTER can effectively leverage these abundant and inexpensive off-domain data when fine-tuning the high-level VLM. The hierarchical design presented in HAMSTER also offers additional advantages through the decoupling of VLM training and low-level action prediction. Specifically, while the higher-level VLM is predicting semantically meaningful trajectories from monocular RGB camera inputs, the lower-level policy models can additionally operate from rich 3D and proprioceptive inputs. In doing so, HAMSTER inherits the semantic reasoning benefits of VLMs along with the 3D reasoning and spatial awareness benefits of 3D policy models [110, 162]. Moreover, the high-level VLM and low-level policy model can be queried at different frequencies

In summary, we study a family of hierarchical VLA models HAMSTERs, where finetuned VLMs are connected to low-level 3D policy models [110, 162]. The 2D paths produced by high-level VLMs serve

[*]Representations similar to 2D paths has been explored in the robot learning literature [115], primarily as a technique for flexible task specification. We refer readers to section 5.2 for a detailed discussion.

[†]For human video, this corresponds to the position of the palm center or fingertips.

as guidance for a low-level policy that operates on rich 3D and proprioceptive inputs, allowing low-level policies to focus on robustly generating precise, spatially-aware actions. In our experiments, we observe an average of 20% improvement in success rate over seven different axes of generalization over OpenVLA [165], which amounts to 50% relative gain, as shown in Table C.3. Since HAMSTER is built on both open-source VLMs and low-level policies, it can serve as a fully open-sourced enabler for the community-building vision-language-action models. It is important to note that while we are certainly not the first to propose hierarchical VLA models [115, 259], we propose the novel insight that this type of hierarchical decomposition allows for these models to make use of abundant off-domain data for improving real-world control. This opens the door to alternative ways of training large vision-language-action models using cheaper and more abundant data sources.

5.2 Related Work

LLMs and VLMs for robotics. Early attempts in leveraging LLMs and VLMs for robotics are through pretrained language [152, 327, 332] and visual [316, 271, 253, 225] representations. However, these are insufficient for complex semantic reasoning and generalization to the open world [39, 422]. Recent research has focused on directly leveraging open world reasoning and generalization capability of LLMs and VLMs, by prompting or fine-tuning them to, e.g., generate plans [88, 146, 199, 194, 332, 40] or construct value [145] and reward functions [177, 334, 396, 222, 369]. Our work is more closely related to VLA models, summarized below.

Monolithic VLA models as language-conditioned robot policies. Monolithic VLA models have been proposed to produce robot actions given task description and image observations directly [39, 154, 422, 347, 165, 292]. Monolithic VLA models are often constructed from VLMs [205, 21, 85, 198], and are trained on large-scale on-robot data [39, 67, 163] to predict actions as text or special tokens. However, due to the lack of coverage in existing robotics datasets, they must be finetuned in-domain on expensive

on-robot data. Their action frequency is also constrained by inference frequency, limiting their capability to achieve dexterous and dynamic tasks. The most relevant monolithic VLA model to our work is LLARVA [266], which predicts end-effector trajectories in addition to robot actions. However, LLARVA only uses trajectory prediction as an auxiliary task to improve the action prediction of a monolithic VLA model. In contrast, our work takes a hierarchical approach, enabling us to use specialist lower-level policies that take in additional inputs the VLMs cannot support, such as 3D pointclouds, to enable better imitation learning. Our predicted paths then enable these lower-level policies to generalize more effectively.

VLMs for predicting intermediate representations. Our work bears connections to prior methods using vision-language models to predict intermediate representations. These methods can be categorized by the choice of predicted representations:

Point-based predictions: A common intermediate prediction interface has been keypoint affordances [336, 338, 261, 398, 172]. Keypoint affordances can be obtained through using open-vocabulary detectors [243], iterative prompting of VLMs [261], or fine-tuning detectors to identify certain parts of an object by semantics [338]. Perhaps most related to our work, [398] finetune a VLM to predict objects of interest as well as free space for placing an object, and [202] propose a mark-based visual prompting procedure to predict keypoint affordances as well as a fixed number of waypoints. As opposed to these, our work finetunes a VLM model to not just predict points but rather entire 2D paths, making it more broadly applicable across robotic tasks.

Trajectory-based predictions: The idea of using trajectory-based task specifications to condition low-level policies was proposed in RT-trajectory [115], largely from the perspective of flexible task specification. This work also briefly discusses the possibility of combining trajectory-conditioned model with trajectory sketches generated by a pre-trained VLM. Complementary to RT-Trajectory, the focus of this work is less on the use of trajectory sketches for task specification, but rather a hierarchical design of VLAs such that the high-level VLM can be fine-tuned with relative cheap and abundant data sources. This could

include data such as action-free videos, or simulation data that look very different from the real world. We show that the emergent generalization capability of VLMs from its web-scale pretraining allows it transfer to test scenarios of interest with considerable visual and semantic variations. While RT-trajectory uses human effort or off-the-shelf pre-trained VLMs to generate trajectories, we show that fine-tuning VLM models on cheap data sources can generate significantly more accurate and generalizable trajectories (see Table. C.2). Moreover, our instantiation of this architecture enables the incorporation of rich 3D and proprioceptive information, as compared to monocular 2D policies [115].

Similarly, the emergence of track-any-point (TAP) models [82, 366] has enabled policies conditioned on object trajectories [397, 381, 27] or points sampled from a fixed grid in the image [371]. While our current formulation focuses on end-effector trajectories, this framework can naturally extend to predicting object trajectories or other motion cues. By leveraging the predictive capabilities of VLMs, such an extension could further enhance the model's ability to generalize across diverse scenarios and improve its capacity for fine-grained motion reasoning.

Leveraging simulation data for training robot policies. There has been extensive work on leveraging simulation for robot learning. Simulation data is popular in reinforcement learning (RL), as RL on real robotic systems is often impractical due to high sample complexity and safety concerns [184, 129, 351]. Recently, simulation has been also exploited to directly generate [101] or bootstrap [231] large-scale datasets for imitation learning, to reduce the amount of expensive robot teleoperation data needed. Our work takes a different approach – using simulation data to finetune a VLM, and showing that VLM is able to transfer the knowledge learned from simulation data to real robot systems, despite considerable visual differences. A related observation is recently made by [398], but they use keypoint affordances as the interface between the VLM and the low-level policy as opposed to more general expressive 2D path representations.

5.3 Background

Imitation Learning via Supervised Learning. Imitation learning trains a policy $\pi_\theta(a \mid s, o, z)$ from expert demonstrations, where s denotes proprioceptive inputs, o includes perceptual observations (e.g., RGB images, depth), and z provides task instructions. Given an expert dataset $\mathcal{D} = \{(s_i, o_i, z_i, a_i)\}_{i=1}^{N}$, the policy is optimized via maximum likelihood estimation, maximizing $\mathbb{E}_{(s_i,o_i,z_i,a_i)\sim\mathcal{D}} \left[\log \pi_\theta \left(a_i \mid s_i, o_i, z_i\right)\right]$. Despite advancements in architectures such as 3D policy representations [111, 162], generalizing to novel semantic or visual variations remains challenging. In this paper, we explore how VLMs can enhance imitation learning models for better generalization.

Vision-Language Models. VLMs [200, 198, 205] are large transformer models [359] that accept both vision and text tokens to generate text responses. They are pre-trained on extensive multimodal datasets [417, 44] and later fine-tuned on high-quality, task-specific data [323, 212]. By tokenizing each modality into a shared space, these models autoregressively produce sequences of text tokens conditioned on an image and prior tokens. In our work, we assume access to such a pre-trained, text-and-image VLM [198, 205], further fine-tuned via a supervised loss that minimizes the negative log-likelihood of the target tokens.

5.4 HAMSTER: Hierarchical Action Models for Robotic Learning

In this work, we examine how VLA models can leverage relatively abundant data and demonstrate cross-domain transfer capabilities, as opposed to relying purely on expensive observation-language-action data collected on a robot. HAMSTER is a family of hierarchical VLA models designed for this purpose, exhibiting generalizable and robust manipulation. It consists of two interconnected models: first, a higher-level VLM that is finetuned on large-scale, off-domain data to produce intermediate 2D path guidance (detailed

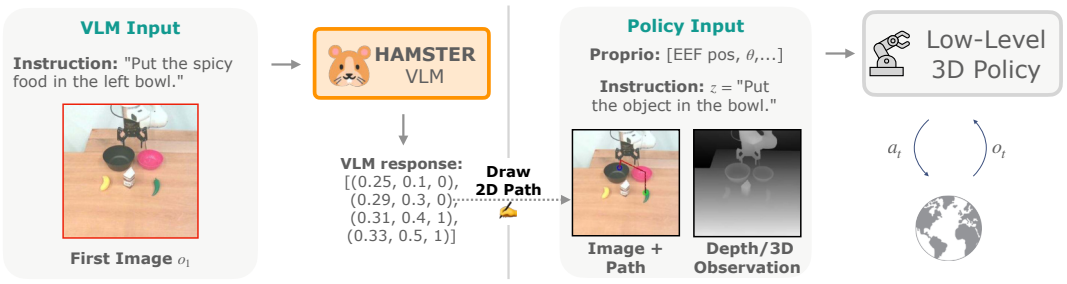


Figure 5.2: Depiction of HAMSTER's execution. The high-level VLM is called once to generate the 2D path. The low-level policy is conditioned on the 2D path and interacts with the environment sequentially to execute low-level actions. The path predicted by the VLM enhances the low-level policy generalization capability.

in Section 5.4.1), and second, a low-level policy that produces actions conditioned on 2D paths (detailed in Section 5.4.2).

The primary advantages of finetuning such a hierarchical VLM that produces intermediate representations as opposed to directly producing actions a with a monolithic model [165, 422, 32] are threefold: *1)* our hierarchical VLM can leverage off-domain datasets lack of precise actions, e.g., simulation and videos; *2)* we find empirically that hierarchical VLMs producing 2D paths generalize more effectively cross-domain than monolithic VLA models; and *3)* the hierarchical design provides more flexibility on the sensory modality, and allows for asynchronous query of large high-level VLA models and small low-level policy models.

5.4.1 HAMSTER's VLM for producing 2D Paths Trained from Off-Domain Data

The high-level VLM of HAMSTER predicts a coarse 2D path p to achieve the task given a monocular RGB image $\texttt{img}$ and language instruction z, i.e., $\hat{p} \sim \texttt{VLM}(\texttt{img}, z)$. The 2D path p describes a coarse trajectory of the robot end-effector, or human hand in the case of human videos, on the input camera image. It also contains information about the gripper state. Formally, the 2D path is defined as $p = [(x_t, y_t, \texttt{gripper\_open}_t)]_t$ where $x_t, y_t \in [0, 1]$ are *normalized pixel locations* of the end effector's (or hand) position at step t, and $\texttt{gripper\_open}_t$ is a binary value indicating the gripper state, i.e., open and close.

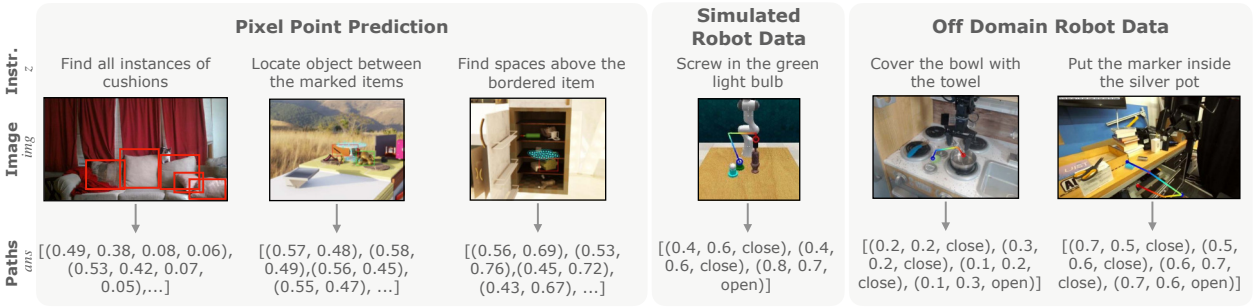


Figure 5.3: **Off Domain Training Data**: $\mathcal{D}_{\text{off}}$ contains (a) Pixel Point Prediction: 770k object location tasks from RoboPoint. (b) Simulated Robot Data: 320k 2D end-effector paths from RLBench environment. (c) Real Robot Data: 110k 2D end-effector paths from Bridge and DROID trajectories.

Although, any pretrained text-and-image-input VLM [198, 205, 3] can be used to predict such a 2D path by casting an appropriate prompt, we find that pre-trained VLMs struggle with predicting such a path in a zero-shot manner (see Table C.2). Therefore, we finetune pre-trained VLMs on datasets that ground VLMs to robot scenes and path predictions collected from easier-to-obtain sources, i.e., internet visual-question-answering data, robot data from other modalities, and simulation data. This is in contrast to work such as [115], where pre-trained VLMs are tasked with directly performing spatially relevant path generation.

We use VILA-1.5-13b [198] as our base VLM, a 13-billion-parameter vision language model trained on interleaved image-text datasets and video captioning data. Although it is possible to curate a dataset on path prediction $\{(\texttt{img}_i, z_i, p_i)\}_i$ and train the VLM *only* on the dataset, the literature [36, 398] has shown that *co-training* the VLM on a variety of relevant tasks, all framed as VQA tasks, can help retain the VLM's generalization capability. To this end, we curate a multi-domain dataset to finetune this model for effective 2D path prediction.

5.4.1.1 Finetuning Objective and Datasets.

Predicting the 2D path of the end-effector requires understanding *what* objects to manipulate in a given task in terms of their pixel positions, but also reasoning about *how* a robot should perform the task. To enable this understanding, we collate a diverse off-domain dataset $\mathcal{D}_{\text{off}}$ from a wide range of modalities, including real-world data, visual question-answering data, and simulation data. Importantly, *none* of this

off-domain data used to train the VLM comes from the deployment environment, thereby emphasizing generalizability.

We assemble a dataset $\mathcal{D}_{\text{off}} = \{(\texttt{img}_i, z_i, \texttt{ans}_i)\}_{i=1}^{M}$ of image inputs $\texttt{img}_i$, language prompts z_i, and answer $\texttt{ans}_i$ consisting of three types of *off-domain* data: (1) pixel point prediction tasks (*what*); (2) simulated robotics tasks (*what and how*); (3) a real robot dataset consisting of trajectories (*what and how*). We detail each dataset below; see Figure 5.3 for visualization of each dataset's prompts and corresponding answers.

Pixel Point Prediction. For pixel point prediction, we use the RoboPoint dataset [398] with 770k pixel point prediction tasks, with most answers represented as a list of 2D points corresponding to locations on the image. A sample consists of a prompt z like `Locate object between the marked items`, an input image `img` and answer `ans` like $[(0.25, 0.11), (0.22, 0.19), (0.53, 0.23)]$.[‡] See the left of Figure 5.3 for an example. This dataset consists of data automatically generated in simulation and collected from existing real-world datasets; its diverse tasks enable the HAMSTER VLM to reason about pixel-object relationships across diverse scenes while retaining its semantic generalization capabilities.

Simulated Robot Data. We additionally generate a dataset of simulated robotics tasks from RLBench [149], a simulator of a Franka robot performing tabletop manipulation for a wide array of both prehensile and non-prehensile tasks. We use the simulator's built-in planning algorithms to automatically generate successful manipulation trajectories. Given a trajectory, we use the first frame from the front camera as the image input `img`. We construct prompt z to instruct the VLM to provide a sequence of points denoting the trajectory of the robot gripper to achieve the given language instruction (see Figure 5.2). The ground-truth 2D path $p = [(x_t, y_t, \texttt{gripper\_open}_t)]_t$ is given by propriceptive projection using forward kinematics and camera parameters.

[‡]Note that this is not a temporally ordered path, but rather a set of unordered points of interest in an image.

We generate 1000 episodes for each of 81 robot manipulation tasks in RLBench, each episode with ∼4 language instructions, for a total of around $320k$ $(\texttt{img}, z, \texttt{ans})$ tuples, where $\texttt{ans} = p$. See the middle of Figure 5.3 for an example.

Real Robot Data. Using real robot data allows us to ensure the VLM can reason about objects and robot gripper paths when conditioned on scenes, including real robot arms. We use existing, online robot datasets *not from the deployment environment* to enable this VLM ability. We source 10k trajectories from the Bridge dataset [362, 67] consisting of a WidowX arm (different embodiment from test robot) performing manipulation tasks and around 45k trajectories from DROID [163]. We covert both datasets to VQA dataset in as similar way as the simulated RL-Bench data, where the 2D paths are extracted from proprioception and camera parameters (see the right of Figure 5.3 for an example). Note that we essentially utilize the robot data as video data, where the end effector is tracked over time. In principle, this could be done with any number of point-tracking methods [82] on raw video as well, with no action or proprioceptive labels.

We finetune the HAMSTER VLM on all three types of data by randomly sampling from all samples in the entire dataset with equal weight. We also include a 660k-sample VQA dataset [204] for co-training to preserve world knowledge. We train with the standardized supervised prediction loss to maximize the log-likelihood of the answers $\texttt{ans}$: $\mathbb{E}_{(\texttt{img}_i, z_i, \texttt{ans}_i) \sim \mathcal{D}_{\text{off}}} \log \text{VLM}\left(\texttt{ans}_i \mid \texttt{img}_i, z_i\right)$.

Remark. One issue with simulation and real robot data is that the extracted 2D paths p can be extremely long, e.g., exceeding one hundred steps. Since we want the HAMSTER VLM to reason at a *high level* instead of on the same scale as the low-level control policy, we simplify the paths p^o with the Ramer-Douglas-Peucker algorithm [295, 83] that reduces curves composed of line segments to similar curves composed of fewer points. We refer readers to Appendix C.7 for an ablation study.

5.4.2 Path Guided Low-Level Policy Learning

The low-level policy of HAMSTER $\pi_\theta(a \mid s, o, z, p)$ is conditioned on proprioceptive and perceptive observations, (optional) language instruction and, importantly, 2D path. While a low-level control policy *can* learn to solve the task without 2D path, the paths allow the low-level policy to forgo long-horizon and semantic reasoning and focus on local and geometric predictions to produce robot actions. As we find empirically (see Figure 5.4), 2D paths allow for considerably improved visual and semantic generalization of low-level policies.

HAMSTER's general path-conditioning framework allows lower-level policies to take in proprioceptive and perceptual (e.g., depth images) observations, that are not input to the high-level VLM. We consider low-level policies based on 3D perceptual information, i.e., $o = (\texttt{img}, \texttt{pointcloud})$, available at test time on a robotic platform with standard depth cameras. We study two choices of policy architecture, RVT-2 [110] and 3D-DA [162] which has shown state-of-the-art results on popular robot manipulation benchmark [149].

Conditioning on Paths. Most policy architectures use the form $\pi_\theta(a \mid s, o, z)$ without 2D path inputs. One naïve option is to concatenate the path with proprioceptive or language inputs. However, because 2D paths vary in length, the architecture must handle variable-length inputs. To incorporate the 2D path $\hat{p}$ from the VLM without major modifications, we alternatively overlay the 2D path onto the image observation [115]. Our implementation follows this approach by drawing colored trajectories on all images in the trajectory $o_i^1, \ldots, o_i^T$: points at each (x_t, y_t) are connected with line segments using a color gradient to indicate temporal progression (see Figure 5.2(b)), and circles mark changes in gripper status (e.g., green for closing, blue for opening). If the policy architecture allows images with more than three channels, we can also include path drawing as separate channels, instead of overlaying it on the RGB channel. We empirically study both drawing strategies, overlay and concatenating channels, in section 5.5.3.

Policy Training. To train the policy, we collect a relatively small-scale task-specific dataset $\mathcal{D} = \{(s_i, o_i, z_i, a_i)\}_{i=1}^{N}$ on the robot hardware. During training, we use *oracle* 2D paths constructed by proprioception projection, similar to how the 2D paths are constructed for the VLM training data, and construct path-labeled dataset $\mathcal{D}_{\text{path}} = \{(s_i, o_i, z_i, p_i, a_i)\}_{i=1}^{N}$. We train a policy $\pi_\theta(a \mid s, o, z, p)$ with standard supervised imitation learning objectives on $\mathcal{D}_{\text{path}}$ to maximize the log-likelihood of the dataset actions: $\mathbb{E}_{(s_i,o_i,z_i,p_i,a_i)\sim\mathcal{D}_{\text{path}}} \log \pi_\theta(a_i \mid s_i, o_i, z_i, p_i)$. For further implementation details, see Appendix C.2.

Inference Speed. Monolithic VLAs query the VLM at every action step [165, 36], which can be very expensive with large VLMs. For example, OpenVLA's 7B-parameter VLA only runs at 6Hz on an RTX 4090 [165]. Instead, HAMSTER's hierarchical design allows us to query the VLM only one or few times during an episode to generate 2D paths $\hat{p}$ that can be followed by low-level policy for multiple steps. Therefore, HAMSTER can be scaled to large VLM backbones without needing end-users to be concerned about inference speed.

5.5 Experimental Evaluation

We evaluate our approach in both simulation and real-world experiments to the following key questions. Do hierarchical VLAs:

Q1 Generalize behaviors to unseen scenarios with significant visual and semantic variation?

Q2 Achieve stronger cross-domain generalization than monolithic architectures?

Q3 Facilitate learning of non-prehensile and long-horizon tasks?

Q4 Exhibit strong demonstration efficiency?

Q5 Have improved visual + semantic reasoning due to hierarchy and VLM fine-tuning?

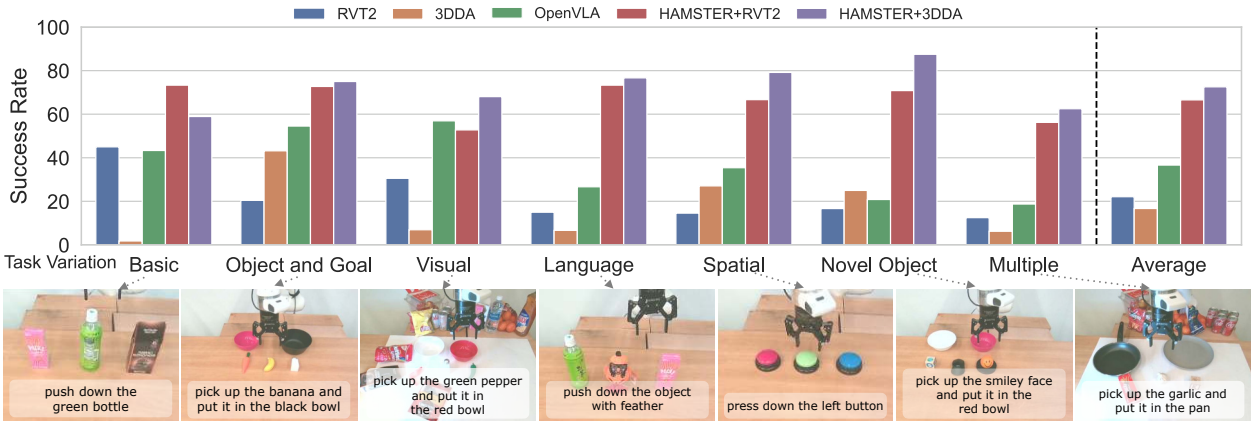


Figure 5.4: Depiction of quantitative real-world policy execution results on a real-world robot, evaluated across different axes of generalization and across both prehensile and non-prehensile tasks. Across all generalization axes, HAMSTER outperforms monolithic VLAs and the base 3D imitation learning policies.

5.5.1 Real World Evaluation on Tabletop Manipulation

To answer Q1, our real-world evaluation experiments aim to test the generalization capability of hierarchical VLA models across significant semantic and visual variations. In particular, we consider a variant of HAMSTER that uses a VLM (VILA-1.5-13b [198]) finetuned on the data mixture in Section 5.4.1 as the high-level predictor, with two low-level 3D policy architectures - RVT-2 [110] and 3D Diffuser Actor (3D-DA) [162] as choices of the low-level policy, as described in Section 5.4.2. The low-level 3D policies are trained with 320 episodes collected via teleoperation shown in Fig. 5.3. Importantly, the high-level VLM has not seen any in-domain data and is only finetuned on the off-domain data described in Section 5.4.1. This suggests that any generalization that the VLM shows result from cross-domain transfer.

Baseline comparisons. To answer Q2, we compare HAMSTER with a state-of-the-art monolithic VLA, OpenVLA [165] as well as non-VLM 3D policies, RVT-2 [110] and 3D-DA [162]. For fair comparison, we finetune OpenVLA on the collected in-domain data described above since OpenVLA showed poor zero-shot generalization. The 3D policy (RVT-2, 3D-DA) baselines are trained with the same teleoperation data used to train the low-level policy in HAMSTER but without the intermediate 2D path representation from HAMSTER's VLM.

Finetuning OpenVLA with RLBench. To ensure our method's advantage over OpenVLA [165] is not solely due to RLBench data, we fine-tuned OpenVLA on the same RLBench dataset used for HAMSTER's VLM—1,000 episodes per task across 81 tasks (using only episodes with good front-camera visibility)—until achieving over 90% token accuracy [165]. We then fine-tuned this model on our tasks following the procedure in Appendix C.3.2. In real-world pick-and-place experiments (6 trials over 6 "Basic" tasks as shown in Table C.1), RLBench-finetuned OpenVLA averaged a success score of 0.54 versus 0.58 for the model without RLBench fine-tuning. This suggests that monolithic VLA architectures like OpenVLA gain little benefit from RLBench data, likely due to mismatches in action and observation spaces relative to the real-world setup.

Quantitative Results. Figure 5.4 summarizes our real-world results. To answer **Q3**, we evaluate across multiple task types, including 'pick and place,' and nonprehensile tasks such as 'press buttons' and 'knock down objects.' We also test generalization across various axes (**Q1**) – *obj and goal:* unseen object-goal combinations; *visual:* visual changes in table texture, lighting, distractor objects; *language:* unseen language instructions (e.g., candy → sweet object); *spatial:* unseen spatial object relationships in the instruction; *novel object:* unseen objects; and lastly, *multiple:* a combination of multiple variations. In total, we evaluate each model on 74 tasks for 222 total evaluations. Detailed results and the success score metric are provided in Appendix Table C.1.

Qualitative Eval on Various Tasks. In addition to the quantitative evaluation conducted for comparison with OpenVLA, we also present qualitative results that demonstrate how HAMSTER's hierarchical structure enables low-level policy models to generalize to more complex tasks. Figure C.1 illustrates the diverse tasks HAMSTER can handle, including unfolding a towel, opening and closing drawers, pressing buttons, wiping surfaces, and cleaning tables. These tasks present challenges such as varying lighting

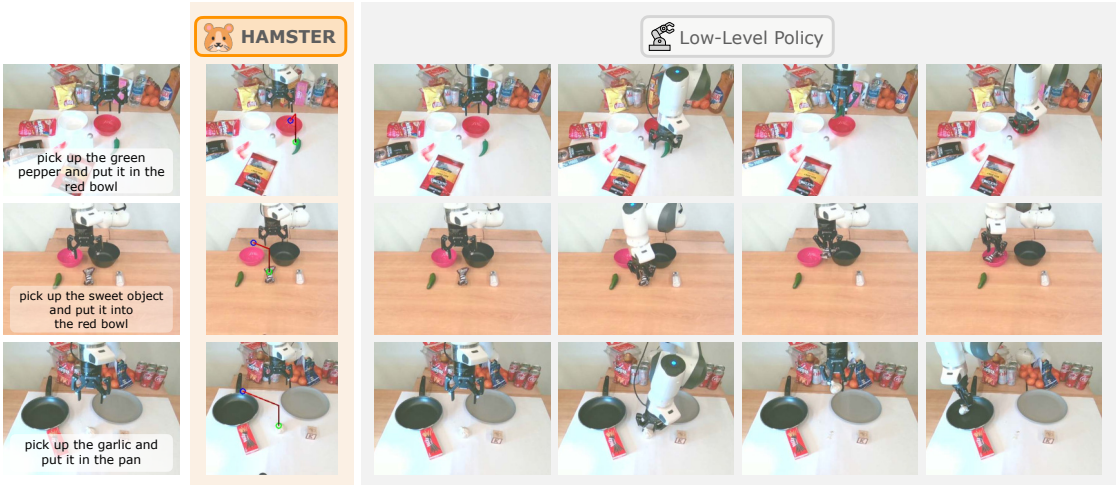


Figure 5.5: Example real-world HAMSTER rollouts demonstrate its strong performance in novel scenes achieved by leveraging VLMs' generalization capabilities and the robust execution of low-level 3D policies.

conditions, cluttered backgrounds, and semantic understanding requiring external world knowledge. Additionally, HAMSTER demonstrates the ability to perform long-horizon tasks—none of which are part of the in-domain training set used to train the policy model.

Overall, we find that HAMSTER significantly outperforms monolithic VLA models and (non-VLM) 3D policies by over **2x** and **3x**, respectively, on average. This is significant because this improved performance is in the face of considerable visual and semantic changes in the test setting, showing the ability of HAMSTER to generalize better than monolithic VLA models or non-VLM base models. We further group results by task type in Table C.3, where we see HAMSTER outperforms OpenVLA across all task types (pick and place, press button, and knock down). See Appendix C.3 for evaluation conditions, a task list, and other experiment details, and Appendix C.5 for failure modes.

5.5.2 Simulation Evaluation

Overall Results. For further investigation into **Q1**, **Q2**, and **Q3**, we conducted a controlled simulation evaluation using Colosseum [288], which provides significant visual and semantic variations across pick-place and non-prehensile tasks. Pairing our high-level VLM with the state-of-the-art 3D-DA [162] policy

Method	Success
3D-DA	0.18 ± 0.10
HAMSTER+3D-DA (50%)	0.36 ± 0.04
HAMSTER+3D-DA	**0.43 ± 0.05**

Table 5.1: Results on Colosseum demonstrate that HAMSTER is data efficient, achieving 2X the success score of 3D-DA with just 50% of the data.

Method	**Original Camera**		**Novel Camera**	
	Success	Complete	Success	Complete
OpenVLA	0.60	0.30	0.23	0.00
HAMSTER+RVT2	0.83	0.70	0.73	0.40
HAMSTER+RVT2 (Concat)	**1.00**	**1.00**	**0.98**	**0.90**

Table 5.2: Real world results demonstrate HAMSTER generalizes to better to novel camera views (see Fig.Figure 5.6). We ran 10 trails and report averaged success rates.

	Avg.	no var	bac tex	cam pos	distractor	lig col	man obj col	man obj siz
3D-DA[Ke, Gkanatsios, and Fragkiadaki]	0.35 ± 0.04	0.43 ± 0.06	0.34 ± 0.07	0.35 ± 0.11	0.39 ± 0.11	0.44 ± 0.13	0.41 ± 0.04	0.41 ± 0.11
HAMSTER (w 3D-DA)	**0.46** ± 0.04	**0.57** ± 0.03	**0.48** ± 0.08	**0.39** ± 0.06	**0.41** ± 0.05	**0.59** ± 0.04	**0.57** ± 0.08	**0.51** ± 0.10
	man obj tex	rec obj col	rec obj siz	rec obj tex	rlb and col	rlb var	tab col	tab tex
3D-DA[Ke, Gkanatsios, and Fragkiadaki]	0.27 ± 0.04	0.34 ± 0.10	0.36 ± 0.05	0.36 ± 0.12	0.07 ± 0.03	0.45 ± 0.12	0.42 ± 0.06	0.23 ± 0.04
HAMSTER (w 3D-DA)	**0.48** ± 0.06	**0.48** ± 0.05	**0.40** ± 0.05	**0.56** ± 0.09	**0.11** ± 0.10	**0.58** ± 0.04	**0.56** ± 0.03	**0.35** ± 0.07

Table 5.3: Simulation evaluation of HAMSTER across different visual variations. We test vanilla 3D Diffuser Actor and HAMSTER across variations in Colosseum [288]. Avg. indicates mean across variations, including no variation.

on RLBench, we compared HAMSTER against a vanilla 3D-DA implementation without path guidance. As shown in Table 5.3 over 5 seeds, HAMSTER outperforms the vanilla approach by an average of 31%. This improvement stems from training with path-drawn images, which encourages the policy to focus on the path rather than extraneous visual features, thereby enhancing robustness to visual variations. We refer readers to Pumacay et al. [288] for details on the variations and Appendix C.6 for further simulation experiment details.

HAMSTER with Fewer Demonstrations. We also test HAMSTER's ability to work well with limited demonstrations to answer **Q4**. We test on a subset of 5 Colosseum tasks, namely, SLIDE_BLOCK_TO_TARGET, PLACE_WINE_AT_RACK_LOCATION, INSERT_ONTO_SQUARE_PEG, STACK_CUPS, SETUP_CHESS. Results in Table 5.1 demonstrate that HAMSTER+3D-DA with just 50% of the data still achieves 2x the success rate of standard 3D-DA, demonstrating that HAMSTER is demonstration-efficient for the downstream imitation learning tasks.

5.5.3 VLM Generalization Studies

Finally, we answer **Q5**: can HAMSTER's hierarchy enable superior visual and semantic reasoning?

Camera View Invariance. We test HAMSTER+RVT2 against OpenVLA from a new camera angle (Figure 5.6) across 10 pick-and-place trials using 6 training objects and 3 training containers to check HAMSTER's visual spatial reasoning. The results in Table 5.2 show that HAMSTER significantly outperforms OpenVLA and remains robust to new camera angles, benefiting from its VLM trained on diverse *off-domain* tasks across various viewpoints. Additionally, we compare HAMSTER+RVT2 (Concat), where instead of overlaying the path on the input RGB image, we modify RVT-2 to accept a 6-channel input by concatenating the original RGB image with a separate RGB image containing only the drawn path. We can easily apply this due to HAMSTER's hierarchical nature. Concatenated paths actually achieve the best performance, demonstrating the effectiveness of this path representation, though it is less general and not compatible with all imitation learning policy architectures (such as 3D-DA as it uses a pre-trained image encoder expecting 3 input channels). One possible explanation is that RVT2's virtual reprojection can fragment the 2D path when it is directly drawn on the image, making it harder for RVT2 to decode. By providing a dedicated path channel (via concatenation), path guidance is preserved more effectively.

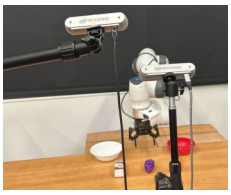


Figure 5.6: Camera pos. for view invariance: old (right) and new (left).

VLM Generalization. We further demonstrate the benefit of HAMSTER's hierarchy by demonstrating that the VLM generalizes well to visually unique and semantically challenging tasks due to its off-domain fine-tuning. We visualize example HAMSTER path drawings in Figure 5.7, demonstrating HAMSTER's VLM itself effectively reasons semantically and visually for unseen tasks. We further investigate VLM performance in Appendix C.4.1, where we find that (1) HAMSTER outperforms zero-shot path generation from closed-source VLMs [115, 194] and (2) that inclusion of simulation data improves HAMSTER's real-world performance. Both results point to the benefit of explicit hierarchy: off-domain VLM fine-tuning that improves its performance. See Appendix C.4.1 for further details.

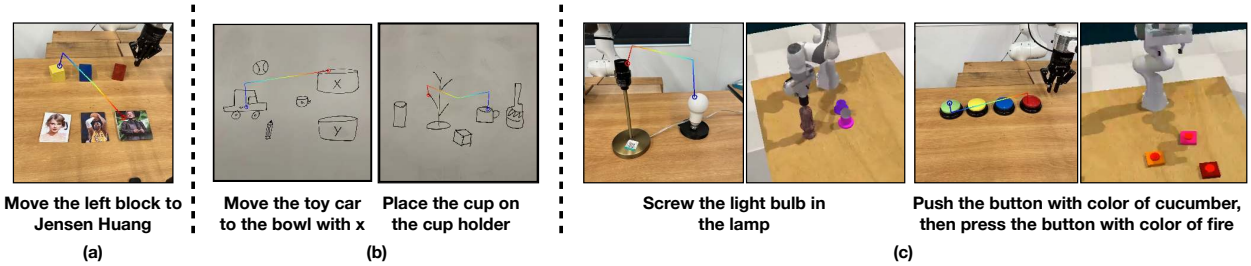


Figure 5.7: HAMSTER's VLM demonstrates strong generalization to unseen scenarios. From left to right: (a) leveraging world knowledge for user-specified tasks, (b) handling out-of-domain inputs like human-drawn sketches, and (c) transferring from diverse simulations to visually distinct real-world tasks. Blue-to-red lines indicate motion, with blue and red circles marking grasp and release points, respectively.

5.6 Conclusion and Limitations

In summary, we study hierarchical VLA models that achieve robust generalization in robotic manipulation. We introduce HAMSTER, consisting of a finetuned VLM that accurately predicts 2D paths and a low-level policy that learns to generate actions using the 2D paths. This two-step architecture enables visual generalization and semantic reasoning across considerable domain shifts while enabling specialist policies, like ones conditioned on 3D inputs, to execute low-level actions.

This work represents an initial step towards developing versatile, hierarchical VLA methods. The proposed work only generates points in 2D space, without making native 3D predictions. This prevents the VLM from having true spatial 3D understanding. Moreover, the interface of just using 2D paths is a bandwidth limited one, which cannot communicate nuances such as force or rotation. In the future, investigating learnable intermediate interfaces is a promising direction. Moreover, training these VLMs directly from large-scale human video datasets would also be promising.

Part II

Adapting to New Scenes and Tasks with Human Guidance

Chapter 6

TAIL: Task-specific Adapters for Imitation Learning with Large Pretrained Models

6.1 Introduction

A desired property of an autonomous agent is the ability to adapt efficiently to novel tasks. In vision and language domains, large pretrained models have demonstrated adaptation to new tasks with just a few examples through prior knowledge obtained from internet-scale datasets [42, 289, 352]. Similar methods have also been applied in decision-making and control applications [39, 84, 36]. However, new control tasks are more difficult to adapt to than the aforementioned vision and language domains due to (1) the lack of internet-scale control data and (2) how optimal actions can vary significantly from task-to-task, even under shared observation spaces. As such, these large-scale decision-making models still rely on a close alignment between training and testing tasks.

In contrast, agents deployed in challenging environments need to adapt to major task variations—take, for example, a general household robot. Equipped with a factory-pretrained policy, the robot will be employed in unique ways by every household. Thus, the robot will need to *continually adapt* in order to best serve each one, e.g., by fine-tuning its capabilities on a few demonstrations [49, 213, 157, 52, 389]. Because most prior decision-making papers adapt to new tasks by fine-tuning the entire model [119, 33, 404, 407,

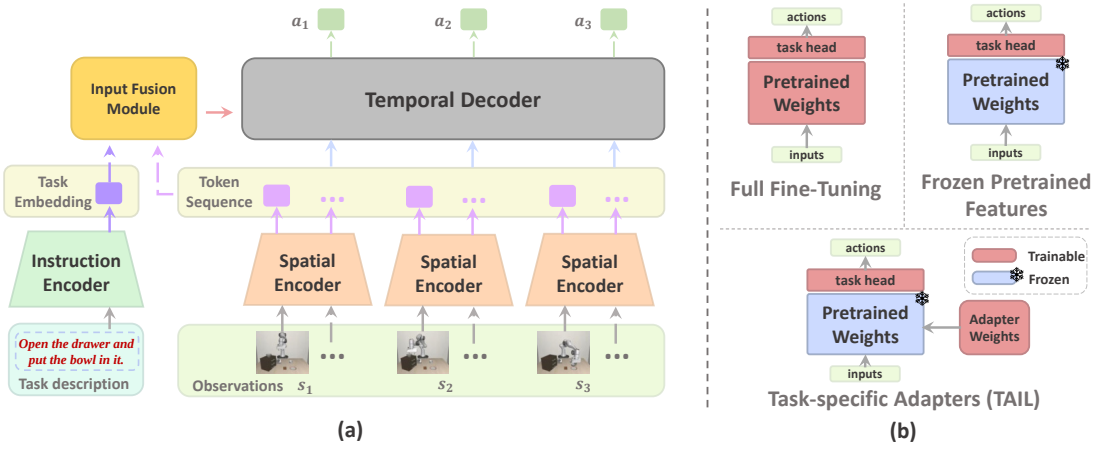


Figure 6.1: **(a)**: The multi-modal, transformer policy architecture we utilize for pretraining. We encode language task descriptions with a pretrained CLIP instruction encoder and image observations with a pretrained CLIP spatial encoder. We additionally encode state observations (not pictured) which, along with the observation embeddings, are embedded into a sequence of tokens used by the temporal decoder transformer to predict single-step action distributions. We include an input fusion module to explicitly combine the task embedding with the observation token sequence for better instruction-following ability. **(b)**: The three types of fine-tuning paradigms we test, with TAIL at the bottom right. For further architecture details, see Appendix Sec. D.1.

67, 209], mastering each new skill requires great computational cost and often leads to catastrophic forgetting of old ones. An alternative approach would be to store a separate policy per new task, which leads to unreasonable storage requirements. Some prior work investigates efficient adaptation of large models to a single task suite [193, 310, 322], but this realistic continual learning setting brings out additional problems to consider, warranting further investigation. What would be the best way for agents to *efficiently adapt* to a stream of novel tasks without having to trade off computation, storage, and performance on older tasks?

To answer this question, we propose **T**ask-specific **A**dapters for **I**mitation **L**earning, shown in Fig. 6.1, a framework for efficient adaptation to new control tasks. Through TAIL we (1) effectively incorporate lightweight adapter modules into pretrained decision-making models and (2) comprehensively compare efficient adaptation techniques implemented in TAIL in a continual imitation learning setting. Notably, we examine parameter-efficient adaptation techniques (PEFT) used for large language models; we explore the potential of adapters [140], prefix tuning [190], and low-rank adaptation (LoRA) [141] in fostering efficient and continual adaptation in large pretrained decision-making models. These works stand out as they introduce a small number of *new* parameters which help: avoid catastrophic forgetting, maintain training

plasticity for continual learning, avoid overfitting with limited adaptation data, and reduce computational and memory burden. Investigating these works in control tasks for a realistic continual learning setup specifically is important because, unlike in language domains, test task losses are often not proportional to test task performance [302, 161]—efficient adaptation insights from language models may not transfer to decision-making ones. Thus, independent investigation of these adaptation techniques for decision-making is crucial for deploying continually adapting agents in the real world.

We compare PEFT techniques implemented in TAIL against commonly used adaptation methods in the imitation learning literature. In our experiments, we discover that TAIL with LoRA leads to the best post-adaptation performance as it preserves the original pretrained representations while being resilient against overfitting in the limited-data regime. These capabilities are especially important for agents operating in new, challenging environments, such as the aforementioned household robots. Our analysis also reveals important insights into the strengths and limitations of each adaptation strategy. Instead of performing full fine-tuning of the entire model, TAIL only introduces a small number of additional parameters without making changes to the original model. These additional parameters make up a mere $\mathbf{1.17}\%$ of the size of the original model. Importantly, this results in approximately $\mathbf{23}\%$ less GPU memory consumption to achieve $\mathbf{22}\%$ higher forward adaptation success rate than full fine-tuning while avoiding catastrophic forgetting. Notably, these results are contrary to many results from the vision and language model literature which show that full fine-tuning works better [132, 235, 55, 310].

In summary, this work bridges a crucial gap in research into efficient and continual adaptation for pretrained decision models by introducing a framework for continual imitation learning, TAIL, and thoroughly analyzing the effects of different efficient adaptation methods. Comprehensive experiments demonstrate that TAIL outperforms standard continual learning and prior single-task adaptation baselines.

6.2 Related Work

Pretrained Models for Control. Researchers have long studied the use of pretrained models for better downstream transfer to related tasks [34, 308, 81]. Recent works have examined using the representations learned by pretrained visual models for control [326, 252, 226, 221, 228]. These methods leverage representations acquired from large task-agnostic datasets, such as Ego4D [112], or through self-supervised objectives. However, there's evidence that simply utilizing these pretrained features may not be as useful for downstream task performance [130]. Meanwhile, another recent line of work directly trains large pretrained models for control [39, 297, 84, 154, 36, 33]. These methods either do not attempt adaptation to new tasks, or perform expensive full-fine-tuning for adaptation. In contrast, our method, TAIL, is a framework for efficient adaptation of decision-making models, like the aforementioned large pretrained control models, and investigates ways to adapt such models efficiently to multiple new tasks.

Parameter-Efficient Fine-Tuning (PEFT). PEFT has gained traction as a way to adapt pretrained models without significantly increasing parameters. Rebuffi, Vedaldi, and Bilen [296] demonstrated that residual adapters for smaller, CNN-based vision models are effective in non-control supervised learning settings. More recently, transformer-focused techniques such as transformer adapter modules [140], LoRA [141], and prompt tuning [190] incorporate lightweight modules or prompts optimized for downstream tasks, all while preserving the original model weights. PEFT offers several advantages over full fine-tuning: it's faster, less susceptible to overfitting, retains prior capabilities, and facilitates efficient task-switching. While PEFT has been successful in both language and vision domains [55, 310], its continuous adaptation for large decision-making models is not yet thoroughly examined. Liang et al. [193] and Sharma et al. [322], Xu et al. [383], and Xu et al. [382] propose the use of adapters, prompt-tuning, and hyper-network in robotics settings, but they do not examine other PEFT methods and focus on adaptation to a single task suite. We instead examine the performance of various state-of-the-art PEFT techniques implemented with TAIL in the *continual learning* scenario.

Continual Learning. Continual learning in control [350, 236, 103] is a long-studied problem with applications to many real-world situations. In general, agents should be able to transfer knowledge (e.g., by continually fine-tuning) or experience (e.g., training data) from previously learned tasks to new tasks [211, 353, 96, 46]. However, with large pretrained models trained on large datasets, fine-tuning the entire model is computationally costly yet risks catastrophic forgetting, and transferring training data from other tasks is too memory inefficient in the face of a large stream of new tasks. Therefore, we present a study into efficient fine-tuning techniques which, when integrated with TAIL, can help inform future research of continual learning.

6.3 Preliminaries

In this section, we introduce our problem setting (Sec. 6.3.1), review large, pretrained models for decision-making (Sec. 6.3.2), and discuss traditional adaptation methods in this area (Sec. 6.3.3).

6.3.1 Continual Imitation Learning

The agent encounters a sequence of K tasks, denoted as $\{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$. Each task $\mathcal{T}_k = (\mu_k^0, g_k)$ is characterized by an initial state distribution μ_k^0 and a goal predicate g_k. Goals for tasks can be specified using language instructions, providing clear context [151, 404]. For every task $\mathcal{T}_k$, the agent receives N demonstration trajectories $\mathcal{D}_k = \{\tau_k^1, \ldots, \tau_k^N\}$. In this paper, we use the standard behavioral cloning loss to optimize the agent's policy π over these demonstrations, however we note that TAIL can be used with other training objectives as well:

$$\hat{\boldsymbol{\theta}} = \min_{\boldsymbol{\theta}} \sum_{k=1}^{K} \mathop{\mathbb{E}}_{s_t, a_t \sim \mathcal{D}_k} \left[\sum_{t=0}^{l_k} \mathcal{L}\left(\pi(a|s_{\leq t}, \mathcal{T}_k; \boldsymbol{\theta}), a_k^t\right) \right]. \tag{6.1}$$

Here, $\mathcal{L}$ is a supervised action prediction (e.g., mean squared error or negative log likelihood) loss, l_k is the length of demonstrations for task $\mathcal{T}_k$, and $\boldsymbol{\theta}$ refers to the *learnable parameters* of the network. Notably, after learning task $\mathcal{T}_k$, the agent cannot access *additional* data from preceding tasks. This presents a continual learning challenge, emphasizing the importance of transferring knowledge across tasks without the risk of catastrophic forgetting [236].

6.3.2 Pretrained Decision-Making Models

Here, we briefly describe common features of large pretrained decision-making model architectures used for embodied agents. We incorporate key components shared amongst these models into the architecture of the model that we pretrain to evaluate efficient adaptation, pictured in Fig. 6.1(a).

Transformer Backbone. Most recent work training large-scale decision-making models [39, 314, 36] utilize a transformer backbone [358] that attends to tokenized observations from prior timesteps. We adopt a standard GPT-2 [290] transformer decoder (Fig. 6.1(a), temporal decoder) with separate encoders for each input modality and continuous action distribution outputs.

Pretrained Input Encoders. Encoders pretrained on large, diverse datasets can produce rich, well-structured embeddings which make it easier to learn the downstream tasks [151, 39]. Therefore, we utilize pretrained CLIP image and textual encoders [289].

Input Modality Fusion. The idea of explicitly "fusing" different input modalities has seen great success not only in domains like vision and language [280], but also in agent learning [151, 39]. Similarly, we utilize FiLM layers [280] (Fig. 6.1(a), input fusion module) to fuse language task specifications with observations.

6.3.3 Adapting pretrained models for new tasks

One standard adaptation method in prior research is full fine-tuning (FFT) of all model parameters (Fig 6.1(b), top left). Though straightforward, it is resource-intensive and prone to overfitting with limited data [33]. There is also a risk of distorting pretrained features, resulting in the loss of prior tasks—a phenomenon known as **catastrophic forgetting** [236]. Evidence also suggests that extensive fine-tuning might undermine a model's rapid adaptability to new tasks, an effect referred to as the loss of **model plasticity and capacity** [173, 215, 176]. Such issues become more prominent in continual learning contexts [211]. Moreover, duplicating a sizable model for each subsequent task is neither efficient nor practical due to storage limitations.

Another standard adaptation method is the use of frozen pretrained features (FPF, Fig 6.1(b) top right). FPF ensures the retention of knowledge acquired from previous tasks by tuning a task-specific head. However, as noted in Sharma et al. [322], it is not expressive enough for out-of-distribution or especially complex tasks. Given these challenges, there's a clear need for a more advanced fine-tuning paradigm that addresses catastrophic forgetting while maintaining model plasticity for adapting to new tasks, all in a data and computationally resource-efficient manner.

6.4 Task-specific adapters for imitation learning

In this section, we outline how we perform efficient adaptation on pretrained models through our **T**ask-specific **A**dapters for **I**mitation **L**earning framework, depicted in Fig 6.1(b). Different from the FPF approach which simply substitutes the policy head for every new task, TAIL introduces a small set of new weights, serving as a lightweight plugin to address specific tasks. This concept draws inspiration from parameter-efficient adaptation techniques prevalent in the language model area. These methods offer several advantages as they: (1) add a few parameters (typically between $0.1\% \sim 2\%$) to preserve the original

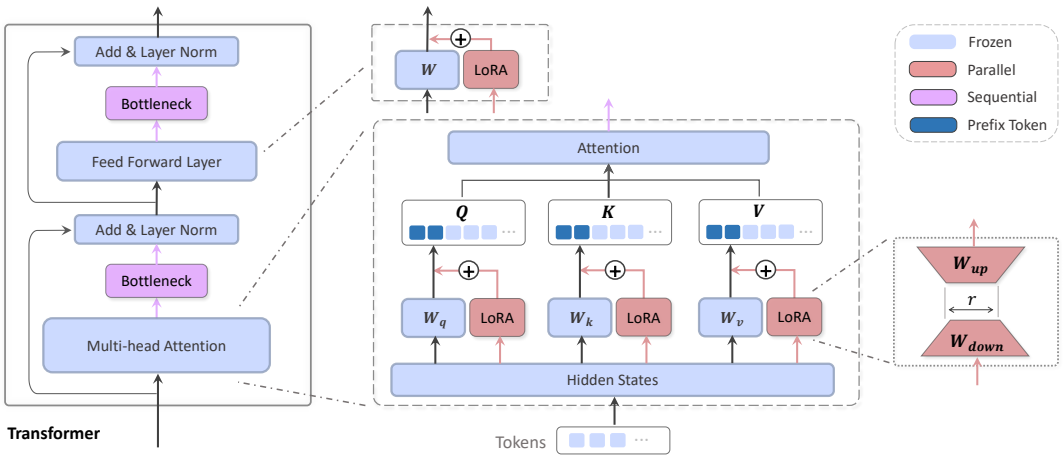


Figure 6.2: Demonstration of three weight integration styles of TAIL for a Transformer block: sequential (bottleneck adapter), parallel (LoRA), and prefix token (prefix/prompt-tuning).

features, thereby enhancing model plasticity for continual learning and avoiding catastrophic forgetting [176], (2) are resilient to overfitting when adaptation data is scarce, (3) are more computationally and storage-efficient than FFT.

Next, we delve into three prominent weight integration techniques for Transformer-based pretrained models in Sec. 6.4.1, followed by a case study illustrating the application of this framework in continual imitation learning scenarios in Sec. 6.4.2.

6.4.1 Adapter Weights Integration

The concept of an adapter can be best conceptualized as a modular plugin to the base model, customized for specific downstream tasks, that does not affect the model's pretrained representations. We mainly explore three prevalent styles of integration for TAIL: **Parallel** [141], **Sequential** [140, 322], and **Prefix Token** [190, 186, 207], all of which are showcased with a Transformer block in Fig. 6.2. Parallel and sequential integration techniques are generally applicable to any model with feedforward layers, while the prefix token style method is especially tailored for Transformers.

Given a pretrained model, let's consider *one* layer weight matrix in it, denoted as $\boldsymbol{W} \in \mathbb{R}^{d\times k}$. Its input and output hidden states are $h_{in} \in \mathbb{R}^d$ and $h_{out} \in \mathbb{R}^k$, respectively. We have $h_{out} = \boldsymbol{W}^\top h_{in}$. Next, we detail how to apply parallel and sequential insertions to the pretrained weight matrix.

Parallel Integration (LoRA). This integration method, often associated with Low-Rank Adaptation (LoRA) [141], introduces trainable low-rank matrices $\boldsymbol{W}_{down} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{W}_{up} \in \mathbb{R}^{r\times k}$. Here, $r \ll \min(d, k)$ represents the rank and is usually much smaller than the dimensions of the original matrix. These matrices are typically integrated in parallel with the original weight matrix $\boldsymbol{W}$ through addition, as shown as LoRA in Fig. 6.2:

$$h_{out} = \boldsymbol{W}^\top h_{in} + \alpha \boldsymbol{W}_{up}^\top \boldsymbol{W}_{down}^\top h_{in}, \tag{6.2}$$

with α being a hyperparameter to modulate task-specific adjustments. The above equation can also be formulated as: $h_{out} = (\boldsymbol{W} + \alpha \boldsymbol{W}_{down}\boldsymbol{W}_{up})^\top h_{in} = (\boldsymbol{W} + \alpha \Delta\boldsymbol{W})^\top h_{in}$, where $\Delta\boldsymbol{W}$ denotes the weight modifications for new tasks, and thus the columns of $\boldsymbol{W}_{down}$ and $\boldsymbol{W}_{up}$ can be interpreted as a new basis that contains task-specific knowledge. As observed by Aghajanyan, Zettlemoyer, and Gupta [6], despite projecting to a condensed subspace with small "intrinsic dimensions," pretrained models can still learn effectively. By introducing the two low-rank matrices, the original weight matrices $\boldsymbol{W}$ can be adeptly tailored with a minimal increase in parameters. Though LoRA was originally crafted for large language models—specifically for the query and value projections matrices W_Q and W_V in multi-head attention [141]—it is easily applied to other linear layers as well, such as the Transformer's feedforward layers [55].

Sequential Integration (Bottleneck Adapter). Renowned in the language model domain, the Bottleneck Adapter introduces bottleneck layers within the model [140, 322] by appending a trainable bottleneck layer after the feedforward network in each Transformer layer. Similar to LoRA, this bottleneck consists of down and up projections, $\boldsymbol{W}_{down}$ and $\boldsymbol{W}_{up}$, which first shrink then restore the dimensions of token

hidden states. Formally, for the feedforward network's input h_{in} and a bottleneck size r, the output h_{out} is:

$$h_{out} = \boldsymbol{W}_{up}^{\top}\phi\left(\boldsymbol{W}_{down}^{\top}(\boldsymbol{W}^{\top}h_{in})\right), \tag{6.3}$$

where ϕ denotes a nonlinear activation function. The Bottleneck Adapter (Fig. 6.2) acts as a filter, isolating relevant information for specific tasks. Yet, filtering often requires a larger bottleneck size compared to that of LoRA, leading to more parameters. Additionally, the sequential insertion can increase latency compared to the parallel nature of LoRA [141].

Prefix Token Integration (Prefix & Prompt-Tuning). In this style, a set of learnable prefix tokens are appended or prepended to the input sequence [190, 186, 207]. Let's consider an input sequence $\mathbf{s} \in \mathbb{R}^{n\times d}$, where n is the sequence length and d is the embedding dimension. The prefix tokens can be represented as $\mathbf{p} \in \mathbb{R}^{m\times d}$, where m denotes the number of prefix tokens. These vectors act like virtual tokens which the original tokens can attend to. They are initialized and learned during the task-specific adaptation phase. The modified input sequence, after appending the prefix tokens, can be expressed as $\mathbf{S} = [\mathbf{p};\mathbf{s}] \in \mathbb{R}^{(m+n)\times d}$. The model then processes this extended sequence. These prefix tokens can be viewed as task descriptors that are designed to guide the model towards the desired task-specific behavior (see Fig. 6.2).

With adapters, we can treat the optimization from Eq. 6.1 as one over adapter weights instead, where the model is parametrized by $\hat{\boldsymbol{\theta}} = \{\boldsymbol{\theta}, \boldsymbol{\omega}\}$ and $\boldsymbol{\omega}$ is the set of adapter weights we are optimizing for.

6.4.2 TAIL for continual imitation learning

We consider the continual imitation learning problem as a typical application of the proposed TAIL adaptation paradigm. The goal of continual imitation learning is to ensure that the model performs effectively on the current task and without significant degradation of performance in past tasks.

Given pretrained model weights, denoted as $\boldsymbol{\theta}$, and a new task $\mathcal{T}_k$ with demonstrations $\mathcal{D}_k = \{\tau_k^1, \ldots, \tau_k^N\}$,

we initialize the task-specific adapter weight $\boldsymbol{\omega}_k$ with far less parameters than the base model: $|\boldsymbol{\omega}_k| \lll |\boldsymbol{\theta}|$. The adapter weights are inserted into the model through the integration methods introduced in Sec. 6.4.1. By optimizing the behavior cloning loss in Eq. 6.1 w.r.t $\boldsymbol{\omega}_k$ while keeping the pretrained weights frozen, the policy adapts to $\mathcal{T}_k$ without interfering with previous tasks.

To execute a task, the corresponding lightweight adapters are loaded as a plugin of the pretrained network weights. For example, when revisiting a prior task T_j, where $j \leq k$, the model is configured to solely activate the j-th adapter $\boldsymbol{\omega}_j$. This entire procedure can be streamlined as follows:

1. For an incoming task $\mathcal{T}_k$, acquire the training set $\mathcal{D}_k$, initialize a task-specific adapter $\boldsymbol{\omega}_k$.

2. Combine adapter $\boldsymbol{\omega}_k$ with the base model $\boldsymbol{\theta}$ using either parallel, sequential, or prefix token.

3. Train the adapter on $\mathcal{D}_k$ to optimize Eq. 6.1 for $\boldsymbol{\omega}_k$, keeping pretrained parameters $\boldsymbol{\theta}$ frozen.

In essence, TAIL ensures task-specific knowledge is contained within the adapters, thereby enabling efficient adaptation without catastrophic forgetting. It's also worth noting that the TAIL framework is flexible. The choice of integration method or the specific architecture of the adapter can be tailored based on the complexity of the task or the available computational resources.

6.5 Experiments

In this section, we evaluate TAIL on a wide range of tasks and benchmark its performance against other fine-tuning approaches. We mainly aim to answer the following questions: (1) Which efficient adaptation methods in TAIL work best? (2) Can TAIL prevent catastrophic forgetting of previously learned tasks, while allowing more efficient forward adaptation to new tasks over standard adaptation methods? (3) What are the computational efficiencies gained by using TAIL? Addressing them requires a set of diverse tasks in realistic environments, as we describe in the following section.

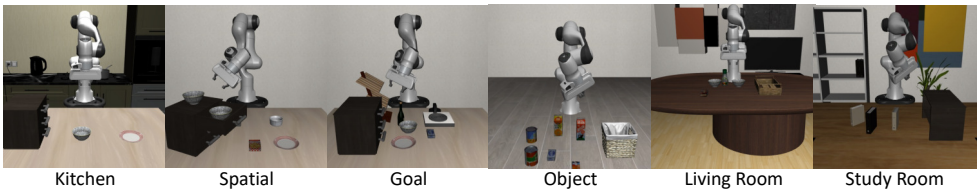


Figure 6.3: Our task suites for continual imitation learning (excluding LIBERO-10). The robot, placed in a tabletop environment, is equipped with a 6-DOF arm and a parallel gripper. It receives RGB images from two views, joint states, and language instructions, and is tasked with producing continuous actions to control its arm.

6.5.1 Datasets and Benchmark Suites

We utilize the LIBERO robotic manipulation continual learning benchmark [201], which features a diverse range of tasks that mirror human daily activities, such as turning on a stove, moving books, and opening drawers. Each task is specified via natural language instructions, for instance, *"Open the top drawer of the cabinet, and put the bowl in it."*

We craft a *pretraining* task suite, named **Kitchen**, involving 40 diverse tasks sourced from the LIBERO-90 dataset's kitchen scenes. We then evaluate *adaptation* to 5 separate task suites. LIBERO contains 3 task suites tailored for continual learning, focusing on evaluating different aspects of knowledge adaptation: the **Spatial** task contains the same objects in each scene but with different spatial layouts; each task in the **Goal** suite has distinct goals (such as open the drawer, or turn on the stove), while keeping the objects and layout fixed; the **Object** suite contains pick-and-place tasks for different objects in the scene but with the same layout. To create a more comprehensive experimental setting, we also create 2 *additional* task suites (from LIBERO-90): **Living Room**, and **Study Room**. We adopt 8 tasks from each of the 5 adaptation task suites, respectively. Finally, we separately evaluate each task sequentially in **LIBERO-10**, a benchmark with 10 challenging long-horizon tasks. See Fig. 6.3 for task suite examples and Appendix Sec. D.4 for more details.

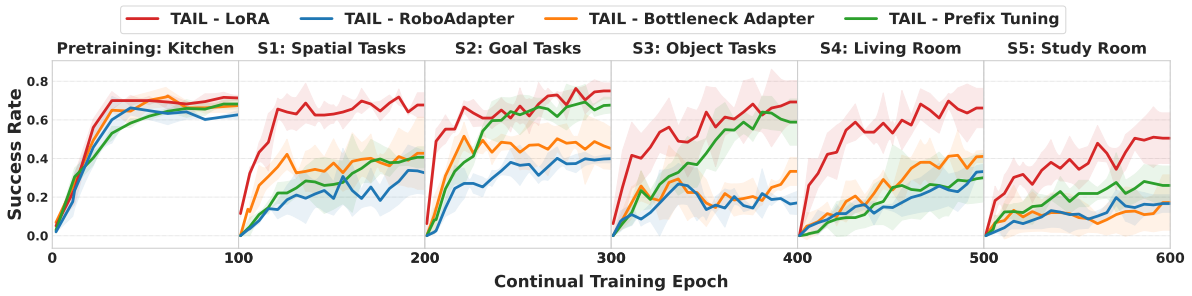


Figure 6.4: Success rates for different types of adapters under our TAIL framework. None of these methods suffer from catastrophic forgetting, so backward evaluation results are not presented here. LoRA performs best across all tasks, underscoring the benefits of the parallel integration approach.

6.5.2 Experiment setup

Evaluation metrics. The primary metric we report is average per-task *suite* success rate, measured by checking if current state aligns with pre-defined goal states. For continual learning, we also assess **Forward Transfer** (FWT) and **Backward Transfer** (BWT) across the curriculum of suites. Following the metric proposed in LIBERO [201], FWT is computed by the maximum success rate one algorithm can achieve when adapting to a new task. We denote FWT at task k as $\mathbf{F}_k$. Meanwhile, BWT measures the success rate increase on previous tasks. Namely, when adapting to the k-th task, we record the best FWT model on this task and then evaluate this model on all previous $k-1$ tasks, obtaining success rate $\mathbf{S}_i, 1 \leq i \leq k-1$. Then we compute the success rate difference between the new model and the best FWT of the previous $k-1$ tasks and then average among them to obtain the BWT metric: $\mathbf{B}_k = \frac{1}{k-1}\sum_{i=1}^{k-1}(\mathbf{S}_i - \mathbf{F}_i)$. For both metrics, higher is better.

Model architecture. We adopt the CLIP-base model [289] as both the spatial encoder and the language instruction encoder, each with 12 transformer layers. A 6-layer GPT2 structure [291] serves as our temporal encoder, with the FiLM module [280] handling input fusion. These components are well-regarded in the literature [54, 39, 154]. Further architectural details can be found in Appendix D.1.

Continual Learning Baselines. We adopt four baselines: Full Fine-Tuning (FFT), Frozen Pretrained Features (FPF) which mirrors the linear probing method [173] but also tunes both the policy head and fusion

module *per task*, Experience Replay (ER) [48] which uses a 50-50 data split between new and previous task data while adapting to a new task [301], Elastic Weight Consolidation (EWC) [169] which regularizes updates of crucial parameters from earlier tasks based on their Fisher information, and PackNet [229] which prunes parameters to then be re-learned for every new task. These all use the same model and task conditioning, i.e., language, as TAIL. Further baseline details in Appendix D.2.1.

TAIL Adapters. We use LoRA [141], Bottleneck Adapter [140], and Prefix Tuning [190] to represent parallel, sequential, and prefix integration styles. RoboAdapter [322], a specific implementation for decision-making, stands as another *sequential* integration style. Unlike the Bottleneck Adapter that applies weights at every transformer layer, it introduces weights only at specific transformer layers and exclusively after the feedforward layer. Configuration specifics and more details for these adapters are available in Appendix D.2.2.

Training, Adaptation, and Evaluation. Each task provides 50 successful human demonstrations. These are divided into 40 training trajectories and 10 for validation. *We report success rates over 10 scenes with initial states that are unseen in training.* This limited demonstration setup offers an opportunity to determine which technique is less prone to overfitting in data-restricted conditions. Given our focus on evaluating the adaptation of large pretrained models, we further increase adaptation difficulty by training on and evaluating adaptation performance on all tasks within a task suite simultaneously.* We pretrain on **Kitchen** until performance convergence (100 epochs). Subsequent adaptations follow two setups: (1) sequential adaptation across the **Spatial**, **Goal**, **Object**, **Living Room**, and **Study Room** task suites for 100 epochs each, and (2) adaptation to each long-horizon task within the **LIBERO-10** benchmark over 50 epochs. Each experiment is conducted with 3 different random seeds. Except for the Experience Replay

*We use one adapter per task *suite*. LIBERO [201] originally evaluated on a per-task basis.

(ER) method, data from earlier tasks remains unavailable in later stages. Our diverse adaptation setup provides a thorough and in-depth examination of knowledge transfer across a spectrum of domains, including spatial, procedural, visual, and compositional.

In the pretraining phase for TAIL, we add trainable adapters to the CLIP spatial and instruction encoders while freezing the encoder weights. All other model weights are fully learnable. During adaptation, the CLIP encoders and the GPT2 decoder are frozen, while adapters for them, the fusion module, and the policy head are tuned. Adapter weights are initialized from previous adapters with minor random noise. A fusion module and policy head copy are maintained during the adaptation for both TAIL and FPF. The detailed hyperparameters are presented in Appendix D.2.

6.5.3 Results and analysis

Comparison of TAIL Integration Styles. Fig. 6.4 showcases the continual adaptation success rates for different TAIL methods. The efficacy of LoRA suggests that a well-pretrained model has a surprisingly low intrinsic dimension for imitation learning tasks [6]. This implies the existence of a low-rank reparameterization that is just as adept for fine-tuning as the full parameter space. Further, the prefix tuning method outperforms the bottleneck-based approach [140], indicating that the sequential integration style may not be the optimal choice for continual learning, potentially due to its inherent "filtering" mechanism. Surprisingly, RoboAdapter [322] generally performs the worst, potentially due to only introducing weights after the feedforward layer as opposed to after [140] or within [190, 141] the attention layer. Due to LoRA's pronounced effectiveness, it is predominantly employed as our TAIL integration method in subsequent experiments.

TAIL vs. Conventional Fine-tuning. Across all evaluations, TAIL vastly outperforms all baselines in both forward and backward transfer, demonstrating that conventional fine-tuning methods are weak in data-scarce continual learning. In Fig. 6.5 we plot continual learning success rates over 6 task suites,

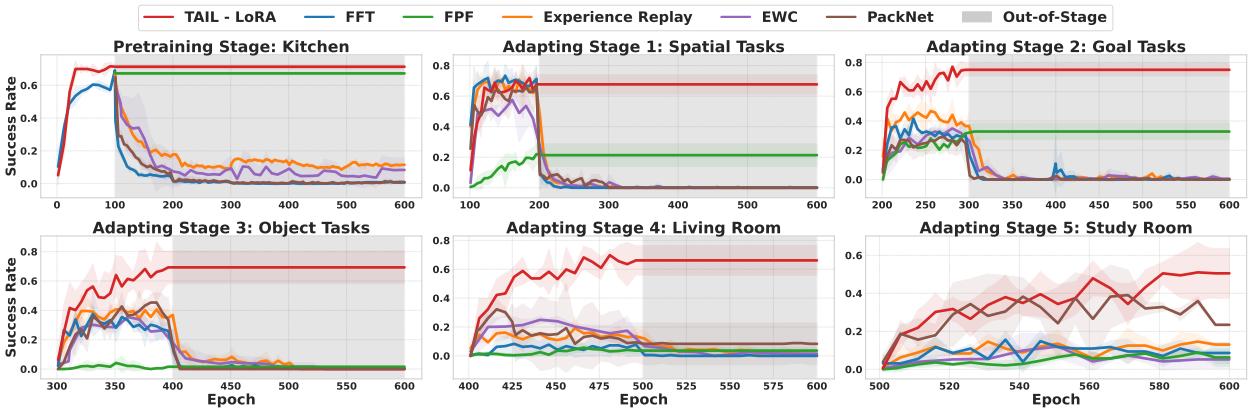


Figure 6.5: Success rates on the pretraining stage on 40 tasks in the LIBERO Kitchen scene and 5 adaptation stages, each with 8 tasks over 100 epochs, which are continuously evaluated in subsequent stages (shaded area).

where TAIL outperforms the best baselines by over **3x** in some comparisons and generally achieves the best success rates. We display additional results on LIBERO-10, long-horizon tasks, in Table 6.1. Here, TAIL again performs best, with perfect backward transfer and forward transfer capabilities significantly better than the baselines: FFT not only exhibits marked catastrophic forgetting of earlier tasks—evidenced by poor BWT—but also compromises the model's adaptability to new tasks. This decline in forward transfer is characterized by a steady descent in success rates as training progresses, displayed in Appendix Table D.4. Such deterioration in flexibility has been recognized in other studies as well [215, 176]. PackNet is able to adapt well on some task suites as it learns new parameters within different parts of the model, but overall is still outperformed by TAIL.

Table 6.1: Adaptation results on 10 long horizon tasks, higher is better. The BWT for TAIL methods are all 0 (no forgetting). FPF results were omitted due to its near-zero performance. See per-task results in Appendix Table D.4.

	Conventional Fine-Tuning Methods						TAIL-based Methods (**Ours**)			
	Full Fine-Tuning		Experience Replay		EWC		LoRA	Prefix	Bottleneck	RoboAdapter
	FWT ↑	BWT ↑	FWT ↑	BWT ↑	FWT ↑	BWT ↑	FWT ↑	FWT ↑	FWT ↑	FWT ↑
Average	0.48 ± 0.10	-0.55 ± 0.21	0.45 ± 0.09	-0.49 ± 0.23	0.30 ± 0.16	-0.43 ± 0.20	**0.70 ± 0.10**	0.51 ± 0.15	0.46 ± 0.11	0.42 ± 0.13

Adaptation Plasticity. Exhaustive fine-tuning on specialized domains has been found to distort pre-trained features [173], undermining model adaptability. Our circle-back experiments in Table 6.2, where

a full fine-tuned model is re-trained on prior task suites, demonstrate a steep performance drop upon re-visiting previously learned tasks. Additional experiments in Appendix D.3.3 further highlight this issue.

The training and validation losses, detailed in Appendix D.3.1 and Fig. D.2, highlight FFT's propensity to overfit. This translates to a notable decline in success rates, reinforcing the challenges FFT faces in balancing retention of prior tasks with the assimilation of new ones.

Table 6.2: The success rate of initial training and revisiting previous tasks with FFT. FFT suffers from catastrophic forgetting and performs worse on re-visits despite re-training on the same data.

Type	LIBERO Task Suite		
	Spatial	Goal	Object
Initial	0.79	0.42	0.42
Re-visit	0.53 (↓ 0.26)	0.20 (↓0.22)	0.27 (↓0.15)

While ER and the regularization-based method EWC exhibit some potential in mitigating catastrophic forgetting, they were detrimental to forward transfer performance. Their downsides are also reflected in storage and computing costs: ER requires more storage for previous data than TAIL LoRA adapter weights (e.g., Kitchen dataset at 28GB vs 7.8MB for TAIL's LoRA adapter). Furthermore, EWC presents significant challenges for larger models because of the increased GPU memory consumption from maintaining a copy of the entire weights of the old model in memory. We also found it to exhibit unstable training due to the regularization loss. More discussions are presented in Appendix D.2.1.

When does TAIL work best? The efficacy of TAIL hinges significantly on the base model's features. We compare TAIL under different pretraining strategies and models in Appendix Sec. D.3.2 and D.3.3. In short, TAIL works best with our pretraining architecture and frozen CLIP visual/language encoders, and performance drops when we fine-tune the pretrained encoders, likely as FFT contaminates the rich CLIP features when fine-tuned in a niche domain with sparse data.

Analysis Summary. We argue in favor of a large pretrained base model augmented with numerous lightweight plugins tailored for different downstream tasks. This framework, TAIL, holds considerable promise for advancing embodied intelligence in real-world applications; the storage footprint of our entire model is about 660MB, and duplicating this model for each task in a stream of oncoming tasks is impractical.

Table 6.3: Comparison of trainable parameters and memory usage for TAIL and FFT. We use (·%) and ↓ (·%) to denote the percentage of trainable parameter and the decrease of GPU memory w.r.t FFT.

Method	Conventional	TAIL-based Methods (Ours)			
	Full Fine-Tuning	LoRA	RoboAdapter	Bottleneck Adapter	Prefix Tuning
CLIP (Spatial & Task Encoder)	149.62M	0.49M	1.29M	1.31M	0.58M
GPT2 (Temporal Encoder)	21.78M	0.69M	0.40M	0.40M	0.24M
Fusion module and policy head	0.84M	0.84M	0.84M	0.84M	0.84M
Total Parameters	172.24M	2.02M (1.17%)	2.53M (1.47%)	2.55M (1.48%)	1.66M (0.93%)
GPU Memory (Batch 14)	20.1G	15.5G (↓ 23%)	14.0G (↓ 30%)	14.9G (↓ 26%)	15.8G (↓ 21%)

Meanwhile, the space occupied by one such model can accommodate as many as 84 task-specific adapters, which, as our experiments show, can outperform full fine-tuning regardless. Moreover, the features of the pretrained weights remain intact, ensuring their applicability across a broad array of domains. In summary, TAIL offers a promising avenue for the efficient adaptation of large decision-making models. Despite the fact that our method requires significantly less computation and memory (and storage), our experiments show that it consistently outperforms all prior approaches in the continual learning setting. We would also like to highlight that the TAIL framework is not restricted to imitation learning, but also other learning methods such as reinforcement learning.

6.6 Conclusion

In this study, we examined the challenges of efficiently adapting large pretrained models for decision-making and robotics applications. We proposed TAIL, an efficient adaptation framework for pretrained decision-making models. Through a comprehensive exploration of parameter-efficient fine-tuning (PEFT) techniques in TAIL, especially Low-Rank Adaptation (LoRA), we demonstrated their potential in enhancing adaptation efficiency, mitigating catastrophic forgetting, and ensuring robust performance across diverse tasks. Our empirical evaluations on the LIBERO benchmark further underscored the advantages of these techniques in continual learning scenarios. As the demand for adaptive, intelligent agents grows across various domains, the insights from this research offer a promising direction for the future of efficient model adaptation in decision-making contexts.

Chapter 7

HAND Me the Data: Fast Robot Adaptation via Hand Path Retrieval

7.1 Introduction

Imagine you want to train a kitchen helper robot to assist with cooking, cleaning, and washing dishes. What is the most efficient way to teach it to perform such a diverse set of tasks? One promising direction is imitation learning on large robotics datasets [67, 163, 412], which has shown to learn capable robot policies [347, 165, 32, 192, 346]. However, these approaches still require significant amounts of task-specific demonstration data for fine-tuning to each new robot embodiment or environment.

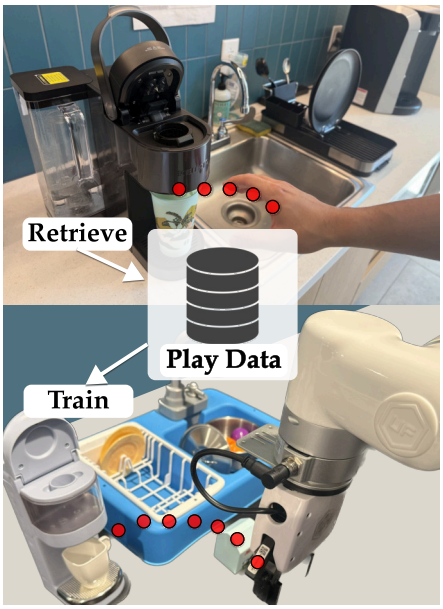


Figure 7.1: HAND learns a policy from as little as one (1) human hand demonstration.

This reliance on *per-task human data collection* makes training a robot that can perform *many tasks* in a specific real-world setting, such as your kitchen, difficult. In contrast, *task-agnostic play data* collected through free-form robot teleoperation [216, 391, 237] is easy to gather, because it does not require constant environment resets or task-specific trajectory labeling. However, play data is difficult to use for training a robot to solve specific downstream tasks without additional labeling. One approach to leverage play data is by retrieving relevant behaviors that can be used

for task-specific training. While prior robot data retrieval methods require additional teleoperated target-task demonstrations [258, 86, 197, 238, 335], we instead propose to retrieve robot data via *human hand demonstrations*, which are easy to provide. Our key insight is to capture *coarse guidance* from the hand demonstration, in the form of relative 2-dimensional hand paths, to retrieve diverse yet relevant behaviors from the play dataset. We propose HAND, a *simple and time-efficient* approach to leverage play data for quickly adapting robots to a range of diverse tasks, requiring as little as *one* human hand demonstration (see Figure 7.1).

HAND avoids the need for calibrated depth cameras [270, 128], specialized eye-in-hand setups [166], or detailed hand-pose estimation [166, 185]. Instead, it first labels a robot play dataset with 2D gripper positions relative to the RGB camera frame by tracking the gripper using a visual point-tracking model [158, 159]. When a human hand demonstration is provided, HAND tracks the hand trajectory with the same simple pipeline. The hand positions are then converted into 2D *relative* sub-trajectories, capturing motion independent of the starting point [402]. After an initial filtering step that removes unrelated behaviors using a visual foundation model [269], HAND retrieves matching sub-trajectories from the play dataset based on the 2D relative hand path. Finally, a policy pre-trained on the full play dataset is fine-tuned on the retrieved sub-trajectories, encouraging the policy to specialize in behaviors relevant to the demonstrated task. In our experiments, we show that because HAND retrieves primarily based on hand motion, it is more robust to irrelevant visual features such as background clutter and lighting changes compared to purely visual retrieval methods.

Our experiments, both in simulation in CALVIN [237] and on a real WidowX robot, demonstrate that HAND enables quick adaptation to **8** diverse downstream tasks with at most **2** provided hand demonstrations. Notably, HAND outperforms the best baseline by **2**$\times$ on a real robot. We also demonstrate HAND works with hand demonstrations collected from completely different scenes from the robot's. Finally,

we also demonstrate that HAND enables *real-time learning* of challenging long-horizon tasks in just **3.5 minutes** of total experiment time while being **5**× faster to collect than teleoperation demos.

7.2 Related Works

Robot Data Retrieval. Prior work has demonstrated *retrieval* as an effective mechanism for extracting relevant on-robot data for training robots [258, 86, 197, 238, 335]. For example, SAILOR [258] and Behavior Retrieval [86] pre-train variational auto-encoders (VAEs) on prior robot images and actions to learn a latent embedding. This latent embedding is used to retrieve states and actions from an offline dataset similar to ones provided in expert demonstration trajectories. However, retrieving based on learned full image encodings or even raw pixel values [335] can be noisy; Flow-Retrieval [197] instead trains a VAE to encode *optical flows* indicating movement of objects and the robot arm in the scene. Similar to Flow-Retrieval, our method HAND also retrieves based on robot arm movement. However, rather than training a dataset-specific VAE model that may not be robust to large visual differences, we retrieve from our offline robot data by primarily matching motions of a human hand demonstration using *relative 2D paths* of the robot end-effector in the prior data. This hand path retrieval helps us robustly retrieve relevant robot arm *behaviors*.

STRAP [238] addresses visual retrieval robustness issues of prior work by using features from DINO-v2 [269], a large pre-trained image-input foundation model for retrieval. However, STRAP, along with all aforementioned retrieval work, assumes access to expert robot demonstrations for the target tasks. HAND on the other hand, only requires a *single*, easier-to-collect human hand demonstration and results in better retrieval and downstream task success rate compared to STRAP.

Learning From Human Hands. Similar to HAND, a separate line of work has proposed methods to use human hands to learn robot policies. One approach is to train models on human video datasets to predict

future object flows [381, 397] or human affordances [20, 172]. These intermediate affordance and flow representations are then used to either train a policy conditioned on this representation [381] on robot data or control a heuristic policy [397, 20, 172]. Other works focus on learning directly from human hands [270, 128, 166, 160, 185]. These works generally use hand-pose detection models aided by multiple cameras or calibrated depth cameras to convert hand poses directly to robot gripper keypoints [270, 128, 185]. However, works that exclusively retrieve human data are restricted to constrained policy representations as they must match human hand poses to robot gripper poses. Kim, Wu, and Finn [166] instead use an eye-in-hand camera mounted on a human demonstrator's forearm to train an imitation learning policy conditioned on robot eye-in-hand camera observations. Unlike these prior works, HAND only requires a single RGB camera from which the robot gripper can be seen. Also, we focus on retrieving robot play data, allowing us to train arbitrarily expressive policies without constrained policy representations [270, 128, 185] or intermediate representations [381, 397, 20, 172].

7.3 HAND: Fast Robot Adaptation via Hand Path Retrieval

We assume access to a dataset of task-agnostic robot play data, $\mathcal{D}_{\text{play}}$, consisting of trajectories $\tau_i = \{(o_t, a_t)\}_{i=1}^{T}$, where each o_t is per-timestep observation that includes RGB images of the robot gripper and robot proprioceptive information, and a_t is the robot action. These trajectories may span many scenes or tasks and can vary in length, potentially covering long-horizon behavior. We do not assume task labels (e.g., language labels), as data collection is easier to scale without labeling each sub-trajectory in a long-horizon play trajectory.* We assume the RGB camera's angle relative to the robot base is fixed across trajectories, which is the case for tabletop robot manipulation setups.

In contrast to prior methods for retrieving robot data [258, 86, 197, 238], we do not assume access to robot demonstrations for the target task.† Instead, for each desired target task, we assume a human

*HAND can also easily incorporate task labels as an extra policy conditioning input.
†Our experiments show that HAND outperforms baselines which have access to expert robot demos.

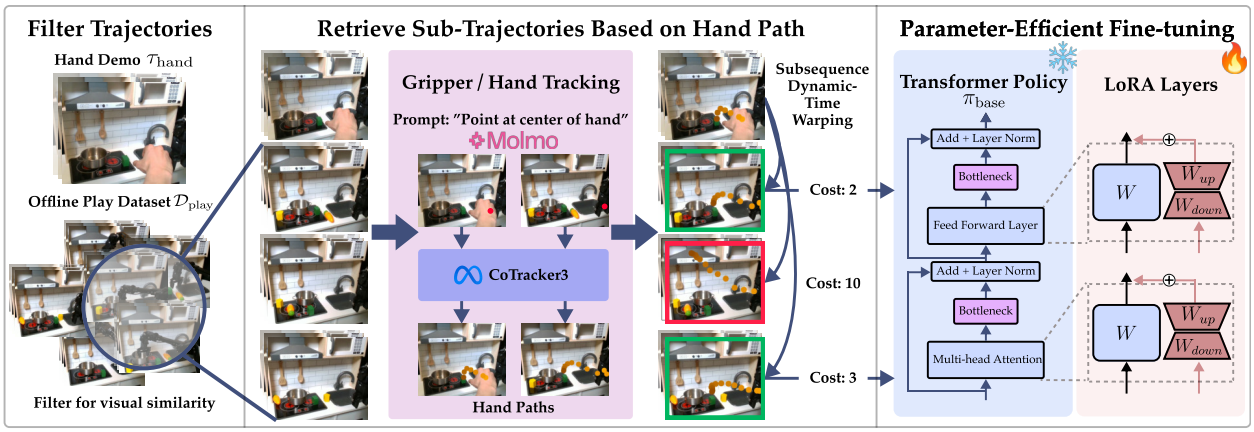


Figure 7.2: **HAND** enables fast-adaptation to a new target task by using an easy-to-provide hand demonstration of the target task (Left). We propose a two-step retrieval procedure where we first filter the trajectories in the offline play dataset, $\mathcal{D}_{\text{play}}$, for visually similar trajectories based on features from a pretrained vision model. We use off-the-shelf, pretrained hand detection and point tracking to construct 2D paths of the motion for both the human hand and robot end-effector. We use these paths as a distance metric to retrieve relevant trajectories from the play dataset (Middle) for quickly fine-tuning a pretrained transformer policy on the target task (Right).

demonstrates it with their hand *without* teleoperating the robot. Our experiments show that these human hand demonstrations in $\mathcal{D}_{\text{hand}}$ can be collected quickly, on average $5\times$ faster than robot teloperation demonstrations, by untrained users. Additionally, high-quality human hand demonstrations may require significantly less effort than high-quality robot teleoperation demonstrations [376, 188]. Each demonstration video in $\mathcal{D}_{\text{hand}}$ consists of a sequence of RGB image observations $o_1, \ldots, o_H$. We assume that the hand videos are captured from approximately the same position relative to the human hand as the robot play data's image observations to the robot gripper.

Given $\mathcal{D}_{\text{play}}$ and $\mathcal{D}_{\text{hand}}$, we aim to train a policy $\pi_\theta(a \mid o)$ to perform the target task demonstrated by the human in $\mathcal{D}_{\text{hand}}$. Since we do not assume task labels in $\mathcal{D}_{\text{play}}$ and we are provided no expert robot teleoperation demonstrations, we must *retrieve* sub-trajectories indicating how to perform the behavior demonstrated in $\mathcal{D}_{\text{hand}}$ from $\mathcal{D}_{\text{play}}$ for training π. We denote this retrieved dataset, which we later use for imitation learning, as $\mathcal{D}_{\text{retrieved}}$. Thus, the key challenges we resolve in our method HAND are: (1) designing a representation that can unify the behaviors in robot sub-trajectories and human hand demonstrations (Section 7.3.1), (2) retrieving relevant sub-trajectories based on a distance metric between these

representations (Section 7.3.2), and (3) time-efficiently training a policy that can perform various unseen target tasks with a high success rate without expert demonstrations (Section 7.3.3). See Figure 7.2 for an overview and Algorithm 7 for full algorithm pseudocode.

7.3.1 Path Distance as a Unifying Representation for Retrieval

Existing robot data retrieval methods assume access to expert demonstrations from which they extract proprioceptive information (e.g., joint states and actions) alongside visual features for retrieval [258, 86, 197, 238, 335]. However, since $\mathcal{D}_{\text{hand}}$ contains only visual data and no robot actions, retrieval based purely on appearance can be noisy—especially due to the visual domain gap between hand demonstrations in $\mathcal{D}_{\text{hand}}$ and robot demonstrations in $\mathcal{D}_{\text{play}}$ (c.f., Figure 7.2, left). To address these issues, we propose an embodiment-agnostic, behavior-centric retrieval metric that enables matching between $\mathcal{D}_{\text{hand}}$ and $\mathcal{D}_{\text{play}}$ based on demonstrated behaviors rather than appearance.

Using 2D Paths for Retrieval. The movement of the robot end-effector over time provides rich information about its behavior [192]. We represent behaviors in both datasets using the paths traced by the human hand or the gripper. Because we assume access only to an RGB camera from which the hand or the gripper is visible (i.e., no depth), we construct these paths in 2D relative to the camera viewpoint for both $\mathcal{D}_{\text{play}}$ and $\mathcal{D}_{\text{hand}}$.[‡]

Obtaining Paths from Data. To extract paths, we use CoTracker3 [159], an off-the-shelf point tracker capable of tracking 2D points across video sequences, even under occlusion. CoTracker3 only requires a single point on the gripper or hand to generate a complete trajectory. We use Molmo-7B [77], an open-source 7B image-to-point foundation model, to automatically select this point by prompting it at the *mid-point* of each trajectory with either "Point at the center of the hand" or "Point to the robot gripper." Using

[‡]If both datasets have additional calibrated depth information, HAND can also operate on 3D paths.

the middle frame ensures a higher chance of visibility in case the gripper or hand is not yet in frame at the beginning or occluded at the end.[§]

Given the 2D point $(x, y)_{\text{hand}}$ or $(x, y)_{\text{play}}$ from the middle frame, we use CoTracker3 to perform bidirectional point tracking, resulting in a 2D path $p_{\text{hand}} = \{(x_t, y_t)_{\text{hand}}\}_{t=1}^{H}$ or $p_{\text{play}} = \{(x_t, y_t)_{\text{play}}\}_{t=1}^{T}$ for each trajectory. See the Gripper/Hand Tracking block of Figure 7.2 for a visualization of this pipeline. Next, we describe how we use 2D paths to retrieve sub-trajectories from $\mathcal{D}_{\text{play}}$.

7.3.2 Retrieving Relevant Sub-Trajectories using Path Distance

Background. For identifying relevant sub-trajectories in $\mathcal{D}_{\text{play}}$, we follow Memmel et al. [238] and use Subsequence Dynamic Time Warping (S-DTW) [247], an algorithm for aligning a shorter sequence to a portion of a longer reference sequence. Given a query sequence $Q = \{q_1, q_2, \ldots, q_H\}$ and a longer reference sequence $R = \{r_1, r_2, \ldots, r_T\}$, where $T > H$, the goal of S-DTW is to find a contiguous subsequence of R that minimizes the total cumulative distance between elements of both sequences. In HAND, the query sequences are the 2D hand demo paths $\{(x_t, y_t)_{\text{hand}}\}_{t=1}^{H}$ and the reference sequences are the 2D paths generated from long-horizon robot play data $\{(x_t, y_t)_{\text{play}}\}_{t=1}^{T}$.

Sub-Trajectory Preprocessing. To preprocess the datasets for S-DTW, we first segment the offline play dataset, $\mathcal{D}_{\text{play}}$, into variable-length sub-trajectories using a simple heuristic based on proprioception proposed in several prior works [327, 238]. In particular, we split the trajectories whenever the acceleration or velocity magnitude (depending on what proprioception data is available) drops below a predefined ϵ value, corresponding to when the teleoperator switches between tasks. We find that this simple heuristic can reasonably segment trajectories into atomic components resembling lower-level primitives. We also split the hand demonstrations evenly into smaller sub-trajectories based on how many subtasks the human operator determined they have completed. After sub-trajectory splitting, we have two sub-trajectory

[§]Points can also be obtained heuristically, e.g., if the robot starts from the same position in each $\mathcal{D}_{\text{play}}$ traj.

datasets, $\mathcal{T}_{\text{hand}} = \{t^i_{1:a}, t^i_{a:b}, \ldots, t^i_{H_i - p_i : H_i} \forall\, \tau^i_{\text{hand}} \in \mathcal{D}_{\text{hand}}\}$ and $\mathcal{T}_{\text{play}} = \{t^i_{1:a}, t^i_{a:b}, \ldots, t^i_{T_i - p_i : T} \forall\, \tau^i_{\text{play}} \in \mathcal{D}_{\text{play}}\}$ where p_i is the length of the last sub-trajectory of trajectory i. Inspired by prior work that proposes to cluster trajectories based on relative embedding differences [402], each sub-trajectory is represented in *relative 2D coordinates*, i.e., $p_t = [x_{t+1} - x_t, y_{t+1} - y_t]$. Relative coordinates ensure invariance based on the starting positions of the hand or gripper so that these starting positions do not influence how trajectories are retrieved.

Visual Filtering. One issue with retrieving sub-trajectories based only on path distance is that different tasks can have similar movement patterns. For example, tasks like "pick up the mug" and "pick up the cube" can appear nearly identical in 2D path space. But, the retrieved trajectories for one task may not benefit learning of the other; since we don't assume task labels in $\mathcal{D}_{\text{play}}$, a policy directly trained on "pick up the cube" retrieved sub-trajectories may still fail to pick up a mug. Therefore, before retrieving sub-trajectories with paths, we first run a visual filtering step to ensure that the sub-trajectories we retrieve will be task-relevant. We use an object-centric visual foundation model, namely DINOv2 [269], to first filter out sub-trajectories performing unrelated tasks with different objects. Specifically, we use the DINOv2 first and final frame embedding differences, representing visual object movement from the first to last frame, between human hand demos and robot play data to filter $\mathcal{T}_{\text{play}}$. We find that using this simple method is sufficient to filter out most irrelevant sub-trajectories. For a given image sequence $o^{\text{hand}}_{1:H}$ from a hand sub-trajectory and image sequence $o^{\text{play}}_{1:T}$ from a robot play sub-trajectory, we define the cost as:

$$\text{C}_{\text{visual}}(o^{\text{hand}}_{1:H}, o^{\text{play}}_{1:T}) = \underbrace{||\text{DINO}(o^{\text{hand}}_1) - \text{DINO}(o^{\text{play}}_1)||^2_2}_{\text{first frame DINO embedding difference}} + \underbrace{||\text{DINO}(o^{\text{hand}}_H) - \text{DINO}(o^{\text{play}}_T)||^2_2}_{\text{last frame DINO embedding difference}}. \tag{7.1}$$

We sort these costs and take the M trajectories with lowest cost as possible retrieval trajectories for each human hand demo sub-trajectory in $\mathcal{T}_{\text{hand}}$. The rest are discarded for those hand demos.

Retrieving Sub-Trajectories. Finally, we then employ S-DTW to match the target sub-trajectories, $\mathcal{T}_{\text{hand}}$, to the set of visually filtered segments $\in \mathcal{T}_{\text{play}}$. Given two sub-trajectories, $t_i \in \mathcal{T}_{\text{play}}$ and $t_j \in \mathcal{T}_{\text{hand}}$, S-DTW returns the cost along with the start and end indices of the subsequence in t_j that minimizes the path cost (see Figure 7.2). We select the K matches from $\mathcal{D}_{\text{play}}$ with the lowest cost to construct our retrieval dataset, $\mathcal{D}_{\text{retrieved}}$.

7.3.3 Putting it All Together: Fast-Adaptation with Parameter-Efficient Policy Fine-tuning

We aim to enable fast, data-efficient learning of the task demonstrated in $\mathcal{D}_{\text{hand}}$. To this end, we first pretrain a task-agnostic base policy π_{base} on $\mathcal{D}_{\text{play}}$ with standard behavior cloning (BC) loss. While our approach is compatible with any policy architecture, we use action-chunked transformer policies [410] due to their suitability for low-parameter fine-tuning and strong performance in long-horizon imitation learning [411, 412, 127, 32].

Adapting to $\mathcal{D}_{\text{retrieved}}$. To rapidly adapt to a new task with minimal data, we leverage parameter-efficient fine-tuning using *task-specific adapters*—small trainable modules that modulate the behavior of the frozen base policy. Adapter-based methods have shown promise in few-shot imitation learning [193, 210], making them ideal for our limited retrieved dataset $\mathcal{D}_{\text{retrieved}}$. Following the findings of Liu et al. [210], we specifically insert LoRA layers [142] into the transformer blocks of π_{base}. These are low-rank trainable matrices (typically 0.1%–2% of the base policy's parameters) inserted between the attention and feedforward layers (see Figure 7.2, LoRA Layers). During fine-tuning, we keep π_{base} frozen and update only the parameters of these LoRA layers, θ, using $\mathcal{D}_{\text{retrieved}}$.

Loss Re-Weighting. While our retrieval mechanism identifies sub-trajectories relevant to the target task, not all will be equally useful. To prioritize the most behaviorally aligned examples, we reweight the BC loss with an exponential term $\in (0, \infty)$ following Advantage-Weighted Regression [279], where each

sub-trajectory is weighted based on its similarity (from S-DTW) to the hand demonstration. Intuitively, this upweights the loss of the most relevant examples in $\mathcal{D}_{\text{retrieved}}$ and conversely downweights those that are less relevant. Finally, because trajectory cost scales vary depending on the task being retrieved and the features being used for S-DTW, we rescale the S-DTW costs $C_{i,\text{path}}$ to a fixed range. For each $\tau_i \in \mathcal{D}_{\text{retrieved}}$, its weight $e^{-C_{i,\text{path}}}$ is scaled to between $[0.01, 100]$, where the normalization term comes from the sum of costs of all trajectories in $\mathcal{D}_{\text{retrieved}}$. Our final training loss is:

$$\mathcal{L}_{\text{BC};\theta} = \frac{1}{|\mathcal{D}_{\text{retrieved}}|} \sum_{\tau_i \in \mathcal{D}_{\text{retrieved}}} \underbrace{\exp(-C_{i,\text{path}})}_{\text{Normalized Weight}} \times \underbrace{(-\log \pi_\theta(a \mid o))}_{\text{BC Loss}}. \tag{7.2}$$

7.4 Experiments

Our aim in the experiments is to study the efficacy of HAND as a robot data retrieval pipeline and evaluate its ability to quickly learn to solve new downstream tasks. To this end, we organize our experiments to answer the following questions, in order:

(Q1) How effective is HAND, using 2D relative paths, in retrieving *task-relevant* behaviors?

(Q2) Does HAND work with hand demonstrations from *unseen scenes*?

(Q3) Does HAND enable learning tasks in *new scenes* in simulation?

(Q4) Can HAND enable *real-time, fast* adaptation on a real robot?

7.4.1 Experimental Setup

We evaluate HAND both in simulation using the CALVIN benchmark [237] and on real-world manipulation tasks with the WidowX-250 robot arm.

CALVIN contains unstructured, teleoperated play data in four tabletop manipulation environments {A,B,C,D}, that share the same set of objects, but have different visual textures and static object locations (e.g., slider, button, switch), shown in Figure E.1 (Left). Because it is infeasible to provide explicit human hand

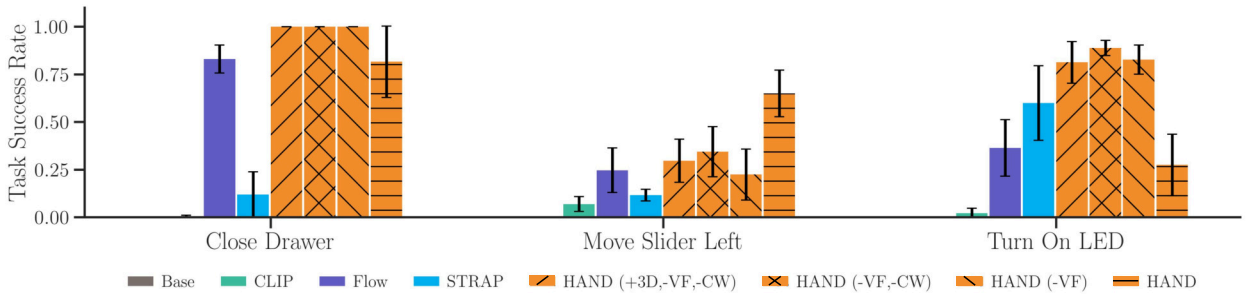


Figure 7.3: **CALVIN Results.** Task success rate of HAND and baseline methods on the CALVIN ABC-D task across three random seeds. Ablations of HAND are denoted by hatches. HAND and ablations outperform the next best baseline Flow on task success rate across all tasks.

demonstrations in CALVIN, we instead perform end-effector point-tracking on expert task demonstrations to mimic the effect of hand-based tracking. We uniformly sample $N = 6$ task-specific expert trajectories from environment D as $\mathcal{D}_{\text{hand}}$, and utilize about 17k trajectories from environments {A,B,C} as $\mathcal{D}_{\text{play}}$. We evaluate our fine-tuned policy in environment D across 3 tasks.

Real World. We demonstrate that HAND can also scale to real-world scenarios by evaluating on several manipulation tasks in a kitchen setup shown in Figure E.2. We collect a task-agnostic play dataset of about 50k transitions. Human teleoperators were instructed to freely interact with the available objects in the scene without being bound to specific task goals. Object positions are randomized within the workspace during data collection and evaluation. We also introduce two difficult, long-horizon tasks, `Put K-Cup in Coffee Machine` and `Blend Carrot`, which require great precision and more than 150 real-world timesteps at a 5hz control frequency to execute, highlighting the capabilities of HAND to learn complex behaviors in real-time. Partial success is provided for tasks composed of multiple subtasks. Refer to Appendix E.1.2 for description of each task.

Baselines: We compare HAND to several retrieval baselines. All methods use the same transformer policy where applicable. We refer the reader to Appendix E.1 for implementation details and Appendix E.5 for extensive ablation results. We consider the following baseline methods:

- $\pi_{\textbf{base}}$ is the base policy pre-trained only on *task-agnostic* play data;

- CLIP retrieves based on cosine similarity between target task language description's CLIP embeddings (instead of hand demonstrations) and play data's CLIP frame embeddings;
- Flow [197] trains a VAE on pre-computed optical flow from GMFlow [380] and retrieves based on latent motion similarity; and
- STRAP [238] also uses S-DTW for sub-trajectory retrieval but computes S-DTW distance based solely on Euclidean distance between DINO-v2 image embeddings.

STRAP and Flow assume access to expert robot demonstrations for both retrieval and fine-tuning. Unless otherwise stated, we adopt them for our setting without expert trajectory fine-tuning because we do not assume access to them. We also perform LoRA fine-tuning; they train from scratch in their original implementations but we found LoRA fine-tuning to perform better for them.

7.4.2 Experimental Evaluation

(Q1): HAND retrieves more task-relevant data. We analyze the quality of retrieved sub-trajectories between Flow, STRAP and HAND. STRAP and HAND use S-DTW-based trajectory retrieval, but STRAP relies purely on visual DINO-v2 embeddings for retrieval. We provide a single hand demonstration of three real robot tasks and retrieve the top $K = 25$ matches from $\mathcal{D}_{\text{play}}$. Compared to STRAP, we observe in Table 7.1 that HAND retrieves more trajectories in which the robot actually performs the hand-demonstrated task. As STRAP retrieves based on visual similarity, it suffers when there is a substantial visual gap between the target demonstrations, e.g., human hand videos, and the offline robot play dataset. In particular,

	Block	Button	Microwave
Flow	7/25	0/25	0/25
STRAP	5/25	0/25	2/25
HAND(-VF)	9/25	13/25	9/25
HAND	**15/25**	**18/25**	**11/25**

Table 7.1: **Number of retrieved sub-trajectories performing demonstrated task.** HAND retrieves more sub-trajectories performing the demonstrated task compared to Flowand STRAP.

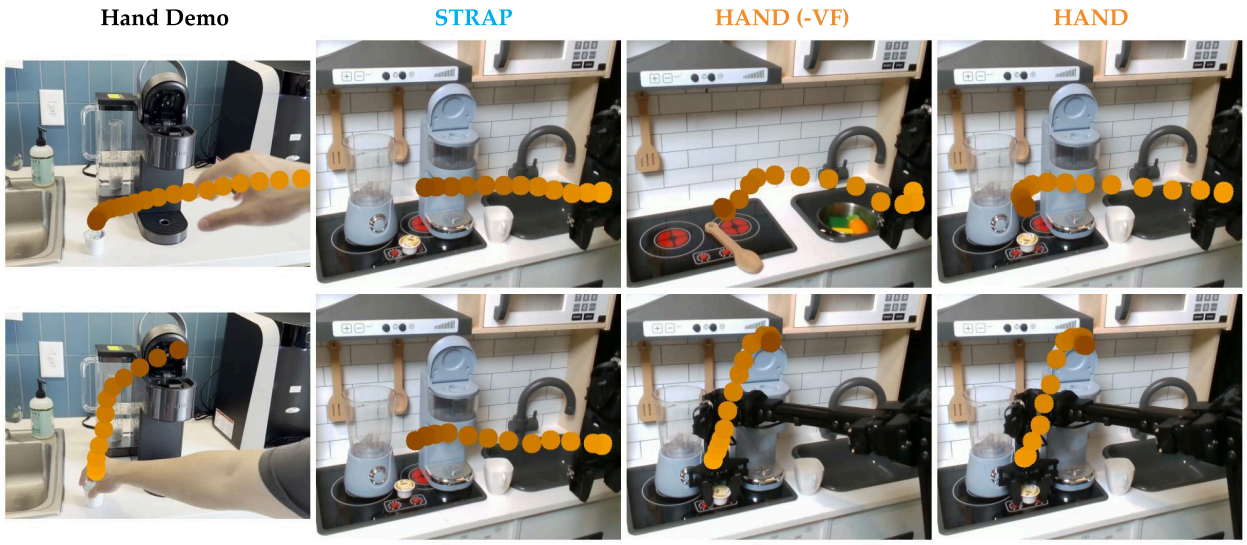


Figure 7.4: **Qualitative retrieval results on out-of-distribution scene.** We visualize the top sub-trajectory match of STRAP HAND without visual filtering (HAND(-VF)), and HAND on two out-of-domain demonstrations recorded from an iPhone camera, showing approaching a K-Cup and putting it into the machine. Only HAND's top match is relevant for both hand demonstrations.

for the Push Button task, STRAP cannot retrieve any button pushing trajectories in its top 25 matches. Moreover, we ablate HAND's visual filtering step and show that it helps retrieve $+30\%$ more relevant trajectories across all tasks. We provide qualitative comparisons of the retrieved trajectories by STRAP and HAND in Appendix E.4.

(Q2): HAND works with hand demonstrations from unseen environments. Because HAND retrieves based on *relative hand motions*, it can work with target hand demonstrations from a completely out-of-distribution scene, provided the camera angle remains relatively close to that in the play dataset. To demonstrate this scene robustness, we collect hand demos from a different scene with a handheld iPhone camera and a real coffee machine. We retrieve from play data containing a completely different scene with a toy coffee machine. In Figure 7.4, we show the lowest cost retrieved sub-trajectory of STRAP compared to HAND and an ablation without the visual filtering step, HAND(-VF). We can see that both trajectories for STRAP and the retrieved trajectory for reaching the coffee cup for HAND(-VF) are irrelevant to

the demonstrated task. By focusing on the *motion* demonstrated by the human hand after *visual filtering*, HAND retrieves more task-relevant trajectories.

(Q3): HAND enables policy learning in simulation and real world. In Figure 7.3, we demonstrate that HAND and ablations outperforms the next-best retrieval baseline across all tasks in CALVIN experiments, highlighting that HAND retrieves trajectories useful for improving downstream performance. In Figure 7.3, we also ablate the use of S-DTW-based loss weighting from Equation (7.2) with HAND(-CW), DINO-v2-based visual filtering from Equation (7.1) with HAND(-VF), and ground truth 3D pose information with HAND(+3D,-VF,-CW), We see that HAND outperforms all of these ablations in Move Slider Left. Surprisingly, in this task, HAND(+3D,-VF,-CW) with priviledged 3D information, even underperforms HAND(-CW). We believe this is because, as HAND(+3D,-VF,-CW) retrieves trajectories based on an exact match in 3D end-effector pose, the retrieved trajectories have little variability and thus fail to generalize to changes in object placement in the scene. In other tasks, we notice that adding visual filtering negatively impacts performance, likely for a similar reason that filtering constrains the diversity of the resulting data subset and as such tuning M is important depending on the task/environment.

Real-world experiments in Figure 7.5 demonstrate that fine-tuning with HAND improves success rates by **+45%** over STRAP across all tasks. Despite visual filtering not always helping in CALVIN, we observe that visual filtering is necessary in real-world experiments to retrieve trajectories where the target object is interacted with, as demonstrated with HAND(-VF)'s worse retrieval performance in Table 7.1. We ablate different K values for real robot tasks in Appendix E.6. We also report the performance of π_{base}, which is trained on all of $\mathcal{D}_{\text{play}}$, as a baseline.

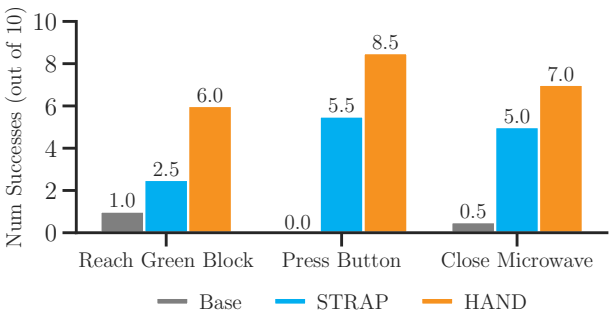


Figure 7.5: **Real-Robot Results.** Number of successes out of 10 of π_{base}, STRAP, and HAND.

(Q4): HAND enables real-time, data-efficient policy learning of long-horizon tasks. We performed two small-scale user studies with IRB approval from our institution to demonstrate real-time learning. In the first study, a participant familiar with HAND iteratively demonstrated each part of a long-horizon `Blend Carrot` task (shown in Figure E.2) and trained a HAND policy ***with over 70% success rate all in under four (4) minutes*** from providing a single hand demonstration to deploying the fine-tuned policy. A full, uncut video of this experiment can be found on our project website.

In the second study, two external users with prior robot teleoperation experience—but not affiliated with this research project—each attempted to collect 10 demonstrations, using both hand and teleoperation methods, to train the robot for the `Put Keurig Cup in Coffee Machine` task (see Figure E.2). We employ HAND retrieval for hand-collected demonstrations and STRAP retrieval for robot teleoperation demonstrations. For a direct comparison, we additionally fine-tune STRAP using the collected teleoperated demonstrations. As reported in Table 7.2, teleoperated demonstrations required over 3× more time to collect than hand demonstrations. Notably, using a single hand demonstration per user, we fine-tuned a policy exceeding 40% success rate compared to STRAP which only achieves 25% with *expert* demonstration. Further increasing the number of expert demonstrations for STRAP to five hurt the downstream performance. We observe qualitatively that adding more expert teleoperated demonstrations reduced the quality of retrievals and thus negatively impacting the downstream policy performance. Our results indicate that hand demonstrations are not only significantly more time-efficient to collect, but that HAND with a single hand demonstration is more effective than STRAP with *multiple expert* demonstrations at learning a new downstream task.

Method	**User 1 (Minutes)**	**User 2 (Minutes)**
Hand Demos (Min) ↓	**3**	**2**
Robot Demos (Min) ↓	10	14
Hand Demos (SR) ↑	**5/10**	**4/10**
Robot Demos (SR) ↑	3/10	2.5/10

Table 7.2: **Hand vs. Robot Teleoperation**. Comparison of time taken and success rates between hand and teleoperated demonstrations.

7.5 Limitations

Relative Camera Viewpoint. One limitation of HAND is that we assume the relative camera viewpoint between the hand demonstration and play trajectories are similar. However, this is a reasonable assumption given that many tabletop manipulation works assume a fixed external camera view. Many open-sourced large-scale offline robot datasets similarly assume standardized camera viewpoints [362, 67, 163, 378]. Moreover, we demonstrated the flexibility of HAND as it is robust to out-of-distribution scenes that are completely different from the ones in the play dataset. In particular, we show that our 2D path retrieval metric is able to retrieve relevant task trajectories even when using a hand demonstration from a regular iPhone camera.

Extending to 3D paths for retrieval. While HAND uses 2D paths for retrieval, one future direction could extend HAND to estimate the hand trajectory in 3D using foundation depth prediction models. Incorporating depth information could provide more fine-grained information about the hand path. Furthermore, 2D hand paths do not provide any explicit information about the gripper for retrieval, which could be useful for more dexterous manipulation tasks. Another direction future work could consider is a mixture of features for improving retrieval for tasks that require more dexterous control, i.e., cloth folding or deformable object manipulation.

7.6 Conclusion

We presented HAND a simple and time-efficient framework for adapting robots to new tasks using easy-to-provide human hand demonstrations. We demonstrated that HAND enables *real-time, unseen* task adaptation with a *single* hand demonstration in just several minutes of policy fine-tuning. Our results highlight the scalability of HAND to train performant real-world, task-specific policies.

Part III

Scalable Adaptation with Minimal Human Supervision

Chapter 8

Bootstrap Your Own Skills: Learning to Solve New Tasks with Large Language Model Guidance

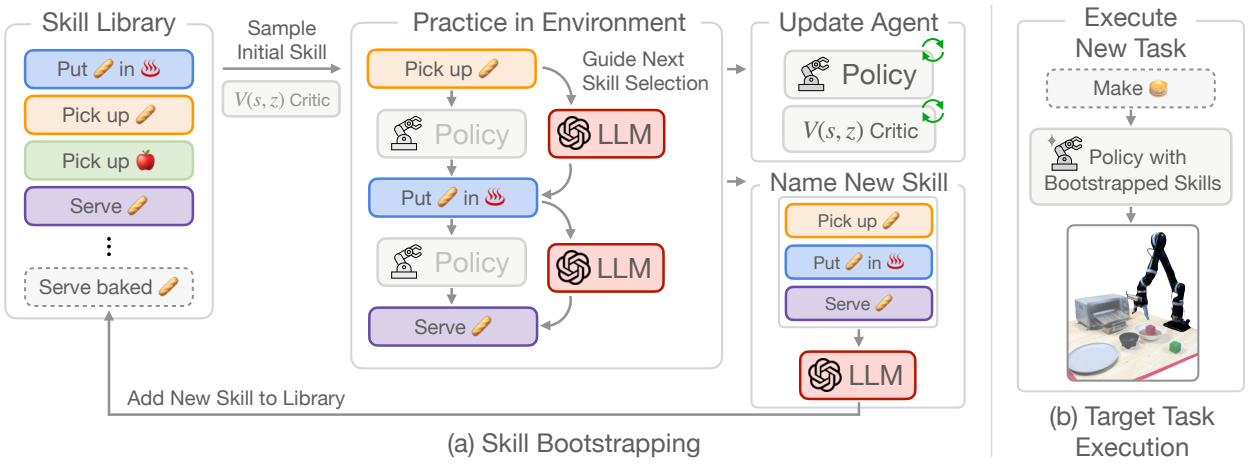


Figure 8.1: BOSS learns to execute a large set of useful, long-horizon skills with minimal supervision by performing LLM-guided *skill bootstrapping*. **(a):** The agent starts with an initial skill library. During bootstrapping, it practices chaining skills into new long-horizon behaviors using guidance from an LLM. The collected experience is used to update the policy. Newly discovered skill chains are summarized with an LLM and added as new skills into the library for further bootstrapping. Thus, the agent's skill repertoire grows over time. **(b):** After bootstrapping, we condition the policy on novel instructions and show execution in the environment using the bootstrapped skill repertoire.

8.1 Introduction

Robot learning aims to equip robots with the capability of learning and adapting to novel scenarios. Popular learning approaches like reinforcement learning (RL) excel at learning short-horizon tasks such as pick-and-place [156, 157, 180], but they require dense supervision (e.g., demonstrations [117, 283, 90, 135] or frequent reward feedback [276, 282, 8]) to acquire long-horizon skills.

In contrast, humans can learn complex tasks with much less supervision—take, for example, the process of learning to play tennis: we may initially practice individual skills like forehand and backhand returns under close supervision of a coach, analogous to RL agents practicing simple pick-place skills using demonstrations or dense rewards. Yet importantly, in between coaching sessions, tennis players return to the tennis court and *practice* to combine the acquired basic skills into long-horizon gameplay without supervision from the coach. This allows them to develop a rich repertoire of tennis-playing skills independently and perform better during their next match.

Can we enable agents to similarly practice and expand their skills without close human supervision? We introduce BOSS (**BO**otstrapping your own **S**kill**S**), a framework for learning a rich repertoire of long-horizon skills with minimal human supervision (see Figure 8.1). Starting from a base set of acquired primitive skills, BOSS performs a *skill bootstrapping phase* in which it progressively grows its skill repertoire by practicing to chain skills into longer-horizon behaviors. BOSS enables us to train generalist agents, starting from a repertoire of only tens of skills, to perform hundreds of long-horizon tasks without additional human supervision.

A crucial question during practice is which skills are meaningful to chain together: randomly chaining tennis moves does not lead to meaningful gameplay; similarly, random chains of pick-place movements do not solve meaningful household tasks. Thus, in BOSS we propose to leverage the rich knowledge captured in large language models (LLMs) to guide skill chaining: given the chain of executed skills so far, the LLM predicts a distribution over meaningful next skills to sample. Importantly, in contrast to existing

approaches that leverage the knowledge captured in LLMs for long-horizon task planning [144, 7, 147, 331], BOSS can use unsupervised environment interactions to *practice* how to chain skills into long-horizon task executions; this practice is crucial especially if the target environment differs from the ones used to train the base skill set. This results in a more robust policy that can compensate for accumulating errors from the initial skill repertoire.

We validate the effectiveness of our proposed approach in simulated household environments from the ALFRED benchmark and on a real robot. Experimental results demonstrate that BOSS can practice effectively with LLM guidance, allowing it to solve long-horizon household tasks in novel environments which prior LLM-based planning and unsupervised exploration approaches fail at.

8.2 Preliminaries and Related Work

Reinforcement Learning Reinforcement learning (RL) algorithms aim to learn a policy $\pi(a|s)$ that maximizes the expected discounted return $\mathbb{E}_{a\sim\pi,P}\left[\sum_t \gamma^t R(s_t, a_t, s_{t+1})\right]$ in a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ represents the transition probability distribution, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ denotes the reward function, and γ is the discount factor. Temporal-difference algorithms are a class of RL algorithms that also learn critic functions, denoted $V^\pi(s)$ or $Q^\pi(s, a)$, which represent future discounted returns when following the policy at state s or after taking action a from state s, respectively [339]. Standard RL algorithms struggle with learning long-horizon tasks and can be prohibitively sample-inefficient.

Skill-based RL To solve long-horizon tasks, prior works have focused on pre-training *skills*, short-horizon behaviors that can be re-combined into long-horizon behaviors [282, 343, 286, 19, 257]. These skills can be represented as learned options [343, 19], sub-goal setting and reaching policies [118, 232], a set of discrete policies [306, 183], or continuous latent spaces that represent behaviors [282, 8, 131, 354,

203]. Yet, most of these approaches need expert supervision (e.g., demonstrations [117, 283, 90, 135, 118, 232, 324], frequent reward feedback [282, 8, 183]). In contrast, BOSS learns to execute long-horizon tasks with minimal human supervision via skill bootstrapping.

Unsupervised RL To learn skills without human supervision, recent works have introduced many unsupervised RL objectives, e.g., based on curiosity [275], contrallability [319, 272], and behavior or state diversification [2, 93, 370, 114, 406]. Because these works learn skills from scratch and explore without supervision, they generally focus on locomotion tasks where most behaviors agents can explore, such as different running gaits, are already meaningful. Few works demonstrate learning of manipulation tasks, but either require hand-crafted state or action spaces [275] or remain constrained to learning simple, short-horizon skills [312, 240]. BOSS makes two improvements to enable bootstrapping of long-horizon tasks: (1) We start from a base repertoire of language-conditioned skills to enable coherent, long-horizon exploration. (2) We leverage an LLM to guide exploration towards meaningful skill-chains within the exponential number of possible long-horizon behaviors.

Language in RL Prior works have employed language to parameterize rich skill sets to train multi-task RL agents [337, 218, 150, 39, 404, 210]. Recent progress in training LLMs has enabled approaches that combine LLMs with pre-trained language-conditioned policies to perform open-loop planning over pre-trained skills [144, 7, 147, 331, 315]. These works do not perform any policy training or finetuning when planning with the LLMs; but instead use the LLMs as top-down planners whose plans are given to fixed low-level skill policies to execute. In contrast, BOSS *pratices* chaining behaviors in the environment during skill bootstrapping and thus learns a more robust, closed-loop policy. This leads to substantially higher success rate for executing long-horizon tasks.

ELLM [87], LMA3 [66], and IMAGINE [64] are closest to our work. ELLM and LMA3 both use an LLM to generate tasks, with the former requiring a captioning model to reward agents and the latter additionally

using the LLM to hindsight label past agent trajectories for task completion; instead, we expand upon a learned skill repertoire, allowing for building skill chains while automatically rewarding the agent based on the completion of skills in the chain. Meanwhile, IMAGINE uses language guidance to generate exploration goals, requiring a "social partner" that modifies the environment according to desired goals. In realistic settings, this social partner requires extensive human effort to design. BOSS instead utilizes LLMs to propose goals in a target environment automatically.

8.3 Method

Our method, BOSS (**BO**otstrapping your own **S**kill**S**), automatically learns to solve new long-horizon, complex tasks by growing a learned skill library with minimal supervision. BOSS consists of two phases: (1) it acquires a base repertoire of skills (Section 8.3.1) and then (2) it practices chaining these skills into long-horizon behaviors in the *skill bootstrapping phase* (Section 8.3.2). BOSS can then *zero-shot execute* novel natural language instructions describing complex long-horizon tasks.

8.3.1 Pre-training a Language-Conditioned Skill Policy

We assume access to a dataset $\mathcal{D}^L = \{\tau_{z_1}, \tau_{z_2}, \tau_{z_3}, ..., \}$ where τ_{z_i} denotes a trajectory of (s, a, s', r) tuples and z_i is a freeform language description of the trajectory. We also assume access to a sparse reward function for the primitive skills, e.g., an object detector that can detect if an object is placed in the correct location. For example, if τ_{z_i} demonstrates a robot arm picking up a mug, then z_i = *"pick up the mug."* and $r = 1$ in the final transition in which the mug is picked up and 0 otherwise. To obtain a language-conditioned primitive skill policy, we train a standard offline RL algorithm on $\mathcal{D}^L$. In our experiments, we use Implicit Q-Learning (IQL) [171] as it is performant and amenable to online fine-tuning. We condition the policy and critic networks on the trajectory's natural language annotation z, yielding a language-conditioned policy $\pi(a|s, z)$ and a critic function $V(s, z)$.

8.3.2 Skill Bootstrapping

After learning the language-conditioned primitive skill policy, we perform skill bootstrapping — the agent practices by interacting with the environment, trying new skill chains, then adding them back into its skill repertoire for further bootstrapping. As a result, the agent learns increasingly long-horizon skills without requiring additional supervision beyond the initial set of skills.

Sampling initial skills. At the start of bootstrapping, the skill repertoire $Z = \{z_1, z_2, ...\}$ is initialized to the set of pre-trained base skills. Upon initializing the agent in the environment at state s_1, we must sample an initial skill. Intuitively, the skill we choose should be executable from s_1 i.e., have a high chance of success. Therefore, in every bootstrapping episode, we sample the initial skill according to probabilities generated from the pre-trained value function, $V(s_1, z)$. We then try to execute the sampled skill until a timeout threshold is reached.

Guiding Skill Chaining via LLMs. If the first skill execution succeeds, the next step is constructing a longer-horizon behavior by chaining together the first skill with a sampled next skill. Naïvely choosing the next skill by, for example, sampling at random will likely result in a behavior that is not useful for downstream tasks. Even worse, the likelihood of picking a *bad* skill chain via random sampling increases linearly with the size of the skill repertoire and *exponentially* with the length of the skill chain. For a modestly sized repertoire with 20 skills and a chain length of 5 there are $20^5 = 3.2M$ possible skill chains, only few of which are likely meaningful.

Thus, instead of randomly sampling subsequent skills, we propose to use large language models (LLMs) to guide skill selection. Prior work has demonstrated that modern LLMs capture relevant information about meaningful skill chains [144, 7, 331]. Yet, in contrast to prior *top-down* LLM planning methods, we explore a *bottom-up* approach to learning long-horizon tasks: by allowing our agent to iteratively sample skill chains and practice their execution in the environment, we train more robust long-horizon task policies

that achieve higher empirical success rates, particularly when generalizing to unseen environments (see Section 8.4).

LLM Prompt Example

Predict the next skill from the following list:

Pick up the mug; Turn on the lamp; Put the mug in the coffee machine; ...

1: Pick up the mug.

2:

Figure 8.2: A shortened LLM prompt. See the full prompt in Appendix F.1.2.

To sample next skills, we prompt the LLM with the current skill repertoire and the chain of skills executed so far. For example, if the agent has just completed *"Pick up the mug"*, we prompt the LLM with the list of skill annotations in Z and then the following prompt: 1. PICK UP THE MUG. 2._____ (see Figure 8.2). The LLM then *proposes* the next skill by generating text following the prompt. We then map this predicted next skill string back to the set of existing skills in Z by finding the nearest neighbor of Z to the proposed skill annotation in the embedding space of a pre-trained sentence embedding model [298]. To encourage diversity in the practiced skill chains, we repeat this process N times and sample the true next skill from the distribution of LLM-assigned token likelihoods. Finally, if the sampled skill is successfully executed, we repeat the same process for sampling the following skill.*

Learning new skills. Once an episode concludes, either because a skill times out or because a defined maximum skill chain length is reached, we add the collected data back into the replay buffer with a sparse reward of 1 for every completed skill. For example, if an attempted skill chain contains a total of 3 skills, then the maximum return of the entire trajectory is 3. We then continue policy training via the same offline RL algorithm used to learn the primitive skills—in our case, IQL [171].

Finally, to maximize data efficiency, we relabel the language instructions for the collected episode upon adding it to the replay buffer. Specifically, following prior work [404], we aggregate consecutive skills into *composite* skill instructions using the same LLM as for skill sampling. We then add the composite

*Note that we do not treat invalid LLM skill chain proposals, like asking the agent to "put keys in a safe" when it has not yet picked any keys up, in a special manner. If the proposal is poor, the agent will fail and the value of the skill will drop with training, making it unlikely to sample the skill chain again.

skill instruction and associated experience to the replay buffer and also add it to our skill repertoire for continued bootstrapping. We store new trajectories with both their lowest level annotations and the LLM-generated composite instructions so the agent can fine-tune its base skills while learning longer-horizon skill chains online. To ensure the agent does not forget its initial skill repertoire, we sample data from the offline dataset $\mathcal{D}^L$ with new data at equal proportions in batch.

In sum, we iterate through these three steps to train a policy during the skill bootstrapping phase: (1) Sampling initial skills using the value function. (2) Sampling next skills by prompting the LLM with skills executed so far. (3) Adding learned skills to the skill library and training on collected agent experience. Algorithm 2 presents a brief overview. The implementation details can be found in Appendix F.2 and Algorithm 8 in Appendix describes the full algorithm.

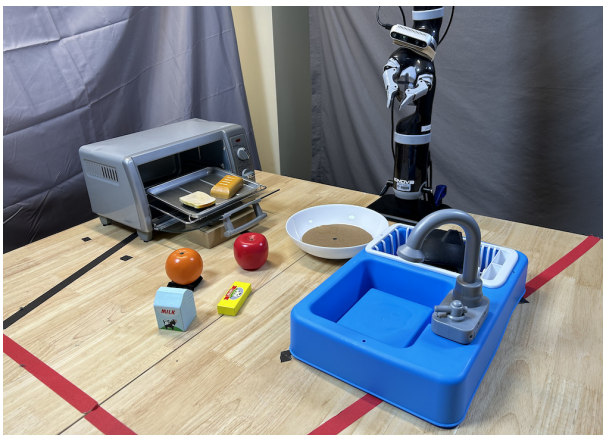
Algorithm 2 BOSS Pseudocode.

```
Train policy π on initial skill repertoire
for skill bootstrapping episode do
    Sample initial skill z and execute
    while not episode timeout do
        Sample next skill from LLM and execute
    Construct composite skill and add to repertoire
    Update policy π
```

8.4 Experimental Evaluation

The goal of our experiments is to test BOSS's ability to acquire long-horizon, complex, and meaningful behaviors. We compare to unsupervised RL and zero-shot planning methods in two challenging, image-based control environments: solving household tasks in the ALFRED simulator [328] and kitchen manipulation tasks with a real-world Jaco robot arm. Concretely, we aim to answer the following questions: (1) Can BOSS learn a rich repertoire of useful skills during skill bootstrapping? (2) How do BOSS's acquired skills

(a) ALFRED benchmark.



(b) Real world Jaco arm setup.

Figure 8.3: **Environments.** (a) **The ALFRED environment** is a benchmark for learning agents that can follow natural language instructions to fulfill household tasks. This illustration was drawn from Shridhar et al. [328] with permission. (b) **Real-world Jaco arm:** Our real-world kitchen manipulation tabletop environment based on RGB image inputs.

compare to skills learned by unsupervised RL methods? (3) Can BOSS directly be applied on real robot hardware?

8.4.1 Experimental Setup

ALFRED Environment. We test our approach in the ALFRED simulator [328] (see Figure 8.3a), since its 100+ floorplans with many interactable objects provide a rich environment for learning numberous long-horizon household tasks. We leverage a modified version of the ALFRED simulator [405] that allows for online RL interactions via a gym interface with 300×300 egocentric RGB image observations. The action space consists of 12 discrete action choices (e.g. turn left, look up, pick up object), along with 82 discrete object types, first proposed by Pashevich, Schmid, and Sun [273]. To train the skills in our initial skill library, we leverage the ALFRED dataset of $73k$ primitive skill demonstrations with language instructions. For bootstrapping we use four unseen floorplans. In each floorplan we define 10 evaluation tasks, each of which requires 2 to 8 primitive skills to complete.

Real-Robot Kitchen Manipulation. We evaluate our method with a real-robot manipulation setup in which a Kinova Jaco 2 robot arm needs to solve stylized kitchen tasks in a table-top environment (see Figure 8.3b). The observations consist of concatenated RGB images from a third-person and a wrist-mounted camera. The robot is controlled with continuous end-effector displacements and discrete gripper open/stay/close commands at a frequency of $10Hz$. To train the initial skills, we collect a dataset of $6k$ language-annotated primitive skill demonstrations via human teleoperation. We perform bootstrapping and evaluate the agents in a table setup with unseen object arrangements.

Training and Evaluation Procedure. We equip the policy with the initial primitive skill library by training it for 150 epochs on the respective pre-collected demonstration datasets using IQL [171] (see Section 8.3.1). We then perform 500,000 and 15,000 steps ($\sim$17 min of robot interaction time) of online skill bootstrapping in the respective unseen eval environments of ALFRED and the real robot setup. Note that for ALFRED we train separate agents for each floorplan, mimicking a scenario in which an agent is dropped into a new household and acquires skills with minimal supervision. After bootstrapping, we evaluate the trained agents *zero-shot* on the held-out evaluation tasks by conditioning the policy on the respective language instruction. To perform well in this evaluation setting, an agent needs to acquire a *large* number of *useful* skills during online environment interactions.

Baselines. We compare BOSS to prior works that can learn a wide range of skills with minimal supervision: (1) unsupervised RL approaches that, like BOSS, learn from environment interactions without additional feedback and (2) large-language model based planners, that leverage the knowledge captured in large pre-trained language models to "bootstrap" given skill libraries into long-horizon behaviors. Concretely, we are comparing to the following approaches:

- **CIC** [178]: SoTA method on the unsupervised RL benchmark [179], expands its skill library with a contrastive alignment objective during bootstrapping. For fair comparison, we pre-train CIC's policy on the same primitive skill dataset used in BOSS before unsupervised bootstrapping.

- **SayCan** [7]: Leverages a pre-trained LLM to break down a given task into step-by-step instructions, i.e., "primitive skills", by ranking skills from a given library. We implement SayCan using the same primitive skill policy pre-trained via offline RL as in BOSS. We use the same LLM as our method, and adapt SayCan's LLM prompt for our environment. Notably, SayCan and similar LLM planning work have no mechanism for fine-tuning to new environments.

- **SayCan+P**: To evaluate the effects of online bootstrapping vs. top-down LLM planning in isolation, we evaluate a SayCan variant that uses *our* LLM-based *skill proposal* mechanism, which leverages the LLM to *generate* step-by-step instructions in place of SayCan's original skill ranking method. We found this to perform better than standard SayCan in our evaluation.

- **SayCan+PF**: SayCan+P on policies fine-tuned in the target environments for the same number of steps as BOSS by sampling single skills with the value function and learning to execute them. This compares the effect of BOSS *learning to chain* skills in the target environments.

Additionally, we evaluate (1) an **Oracle** that finetunes the pre-trained primitive skill policy directly on the target tasks, serving as an upper bound, and (2) a pre-trained primitive skill policy *without any* bootstrapping (**No Bootstrap**), serving as a performance lower bound.

All methods utilize the same base primitive skill policy pre-trained on the same demonstration data. We implement a transformer policy and critic architecture based on Pashevich, Schmid, and Sun [273] trained with the IQL algorithm [171]. All results reported are inter-quartile means and standard deviations over 5 seeds [4]. Finally, Saycan and BOSS all use the LLaMA-13b open-source, 13-billion parameter LLM [352]. For more baseline implementation and training details, see Appendix F.2.

8.4.2 BOSS Bootstrapping Learns Useful Skills

ALFRED. Overall, BOSS achieves superior performance to all non-oracle baselines, with better oracle-normalized return at longer, length 3 and 4 tasks than the best baselines, and BOSS is the **only** method to achieve non-zero success rates across all lengths of tasks. From Table 8.1, the gap between BOSS and best baselines is largest on the length 4 tasks, indicating the benefit of BOSS' LLM-guided skill bootstrapping in learning difficult, longer-horizon tasks without task supervision. CIC can make some progress in some length 3 and 4 tasks, but its contrastive objective generally fails to finetune the primitive skills into meaningful long-horizon skills. Saycan+P performs better than Saycan, indicating that our proposal mechanism better extracts a more meaningful distribution of skills from an LLM, but even Saycan+P greatly falls short of BOSS' performance as it is not robust to execution failures incurred from directly using the pre-trained policy in unseen floor plans. Saycan+PF performs better as it first fine-tunes its policies, but it still achieves a 0% success rate compared to BOSS' 57%. Additional analyses we perform in Appendix F.3.1 demonstrates that in SayCan+P, 95.8% of all unsuccessful SayCan+P trajectories are caused by policy execution failures. SayCan+PF is only slightly better: 95.0% are caused by policy execution failures, indicating that naïve fine-tuning in the target environment is ineffective for solving long-horizon tasks. Since BOSS learns to finetune individual primitive skills and transition *between* skills using a closed-loop policy, it performs much better on complex, long-horizon language-specified tasks in unseen environments.

Table 8.1: Inter-quartile means (IQMs) and standard deviations of oracle-normalized returns, i.e., number of solved subtasks, broken down by task length, across the ALFRED evaluation tasks. We also report oracle-normalized success rate in the last column. We do not report results for length 6 and 8 tasks since not even the oracle was able to learn these.

	Returns by Evaluation Task Length			Average	
Method	Length 2	Length 3	Length 4	Return	Success
No Bootstrap	0.03 +- 0.02	0.05 +- 0.07	0.08 +- 0.09	0.03 +- 0.01	0.00 +- 0.00
CIC [178]	0.02 +- 0.02	0.25 +- 0.08	0.18 +- 0.07	0.11 +- 0.01	0.00 +- 0.00
SayCan [7]	0.06 +- 0.02	0.14 +- 0.00	0.10 +- 0.12	0.06 +- 0.00	0.00 +- 0.00
SayCan + P	0.08 +- 0.04	0.28 +- 0.00	0.20 +- 0.15	0.12 +- 0.01	0.00 +- 0.00
SayCan + PF	**0.64 +- 0.06**	**0.49 +- 0.20**	0.59 +- 0.02	**0.57 +- 0.05**	0.00 +- 0.00
BOSS (ours)	**0.47 +- 0.12**	**0.59 +- 0.13**	**0.81 +- 0.13**	**0.57 +- 0.06**	**0.57 +- 0.14**

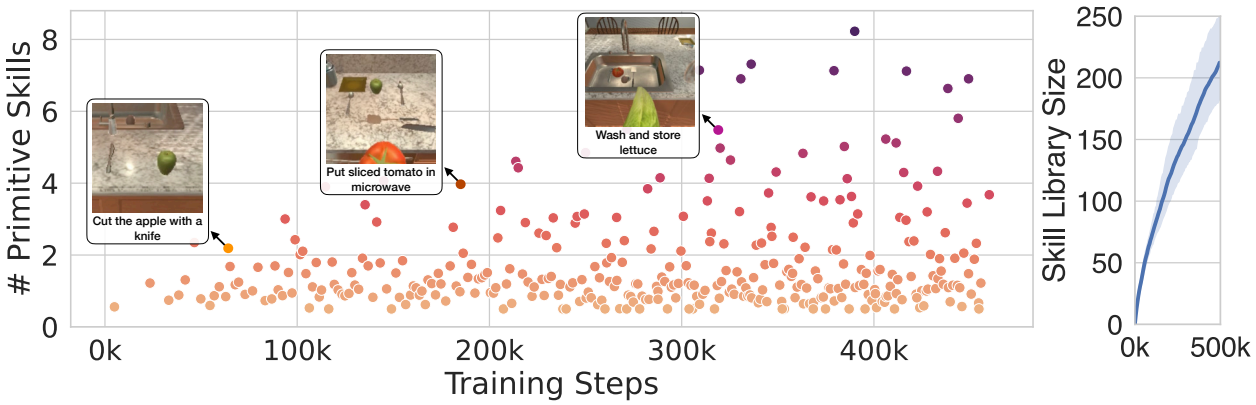


Figure 8.4: **Left**: The number of subtasks in skills executed *during* skill bootstrapping by BOSS in one of the unseen ALFRED floorplans. BOSS progressively learns longer skill chains throughout the course of training. **Right**: The number of newly acquired skills by BOSS throughout training.

We display qualitative examples of a length 2 and 3 task in appendix Figure F.4, where we can see that BOSS successfully completes the tasks whereas Saycan suffers from execution failures, getting stuck while attempting to manipulate objects, and CIC navigates around performing random behaviors (Figure F.4a) or gets stuck navigating around objects (Figure F.4b). We show qualitative examples of learned skills in Figure 8.5 and perform additional experiments and analysis in Appendix F.3.1.

Real Robot. In our real world experiments, we compare BOSS to **ProgPrompt** [331], a similar LLM planning method to Saycan that has been extensively evaluated on real-world tabletop robot manipulation environments similar to ours. We also augment it with prompt examples similar to ours and our skill proposal mechanism. Here, we evaluate on 4 tasks, 2 of length 2 and 2 of length 4 after performing bootstrapping. Results in Table 8.2 demonstrate that both methods perform similarly on length 2 tasks, but only BOSS achieves nonzero success rate on more difficult length 4 tasks as it is able to learn to chain together long-horizon skills in the new environment. See Appendix F.3.2 for more detailed task information.

Table 8.2: Success rates, split by task length, across the 4 robot eval tasks in an unseen table arrangement.

	Evaluation Task Length	
Method	Length 2	Length 4
ProgPrompt [331]	0.65 +- 0.15	0.00 +- 0.00
BOSS (ours)	0.50 +- 0.30	**0.15 +- 0.05**

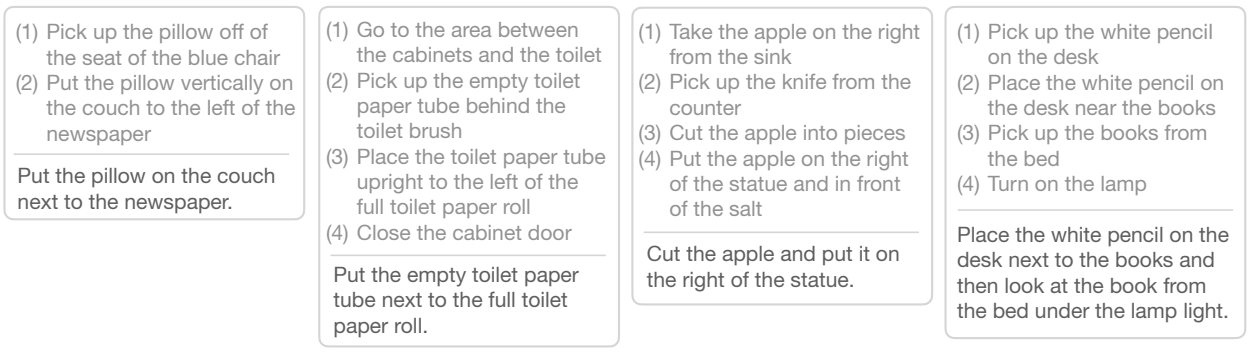


Figure 8.5: Example skill chains (light gray) and new skill summaries (dark grey) learned by BOSS during skill bootstrapping. LLM-guidance ensures meaningful skill chains and summaries.

8.4.2.1 Ablation Studies

To better analyze the effect of our core contribution, the usage of LLM guidance during skill bootstrapping, we compare to the following variants of our approach:

- **BOSS-OPT1**: BOSS bootstrapping with a weaker 1-billion parameter LLM, OPT-1 [408].
- **BOSS-Rand**: An ablation of our approach BOSS that uses *no* LLM guidance during skill bootstrapping and simply selects the next skill at random from the current skill library.

We report results in Table 8.3. The analysis shows the importance of accurate LLM guidance during skill bootstrapping for learning useful skills. Using an LLM with lower performance (OPT1) results in degraded overall performance. Yet, bootstrapping without any LLM guidance performs even worse. Interestingly, the performance gap between BOSS and its variants widens for longer task lengths. Intuitively, the longer the task, the more possible other, less useful tasks of the same length could be learned by the agent during bootstrapping. Thus, particularly for long tasks accurate LLM guidance is helpful.

Table 8.3: ALFRED ablation returns.

	Evaluation Task Length			
Method	Length 2	Length 3	Length 4	Average
BOSS (ours)	0.47 +- 0.12	0.59 +- 0.13	0.81 +- 0.13	**0.57 +- 0.06**
BOSS-OPT1	0.39 +- 0.08	0.36 +- 0.07	0.56 +- 0.08	0.49 +- 0.07
BOSS-Rand	0.32 +- 0.03	0.29 +- 0.11	0.61 +- 0.16	0.43 +- 0.06

To further analyze this, we compare the sizes of the learned skill libraries between BOSS bootstrapped with LLaMA-13B guidance vs. random skill selection (BOSS-Rand) in Figure 8.6. Perhaps surprisingly, the random skill chaining ablation learns *more* skills than BOSS – its skill library grows faster during bootstrapping. Yet, Table 8.3 shows that it has lower performance. This indicates, that while BOSS-Rand learns many skills, it learns less *meaningful* skills. A qualitative analysis supports this intuition: many of the learned skills contain repetitions and meaningless skill chains. This underlines the importance of LLM guidance during skill bootstrapping. Furthermore, the positive correlation between the powerfulness of the used guidance LLM (1B $\rightarrow$ 13B parameters) and the evaluation task performance suggests that future, even more powerful LLMs can lead to even better skill bootstrapping.

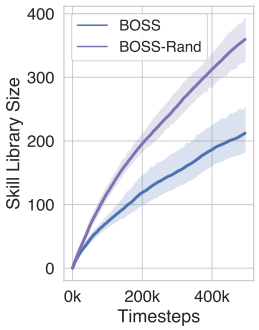


Figure 8.6: Skill library size during bootstrapping.

8.5 Discussion

We propose BOSS, an approach that learns a diverse set of long-horizon tasks with minimal supervision via LLM-guided skill bootstrapping. Starting from an initial library of skills, BOSS acquires new behaviors by practicing to chain skills while using LLMs to guide skill selection. We demonstrate in a complex household simulator and real robot manipulation tasks that BOSS can learn more useful skills during bootstrapping than prior methods.

Limitations. While BOSS learns a large repertoire of skills with minimal supervision, it still has limitations that prevent it from truly fulfilling the vision of agents autonomously acquiring skills in new environments. BOSS requires environment resets between bootstrapping episodes, which are currently performed by a human in our real world experiments. Also, we require success detection for each of the primitive skills during bootstrapping. Future research can investigate using advances in reset-free RL [120, 321] to approach the goal of truly autonomous skill learning. Furthermore, BOSS greedily proposes new

skill chains one skill at a time, this greedy skill chaining process may not be optimal for generating consistent long-horizon behaviors beyond a certain length. In future work, we plan to explore mechanisms to propose long-horizon tasks that are broken down to individual skills in conjunction with the greedy skill chaining of BOSS. Finally, BOSS is currently limited to skills that are combinations of skills in its initial skill library. Extending our work with unsupervised RL [320, 178] techniques for learning new *low-level* skills is an exciting direction for future work.

Chapter 9

RoboCLIP:One Demonstration is Enough to Learn Robot Policies

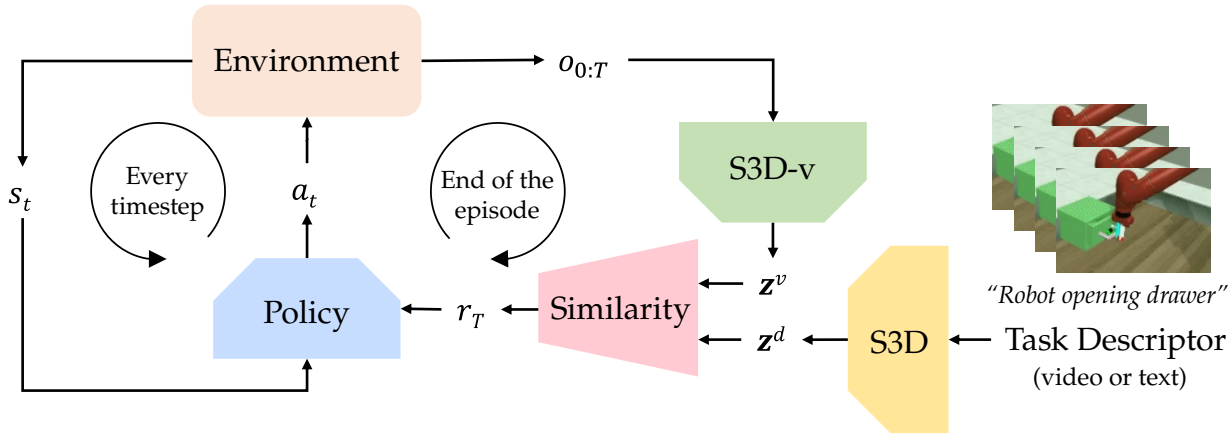


Figure 9.1: **RoboCLIP: Method Overview.** A Pretrained Video-and-Language Model is used to generate rewards via the similarity score between the encoding of an episode of interaction of an agent in its environment, $\boldsymbol{b}z^v$ with the encoding of a task specifier $\boldsymbol{b}z^d$ such as a textual description of the task or a video demonstrating a successful trajectory. The similarity score between the latent vectors is provided as reward to the agent.

9.1 Introduction

Sequential decision-making problems typically require significant human supervision and data. In the context of online reinforcement learning [339], this manifests in the design of good reward functions that map transitions to scalar rewards [11, 125]. Extant approaches to manual reward function definition are not very principled and defining rewards for complex long-horizon problems is often an art requiring significant human expertise. Additionally, evaluating reward functions often requires knowledge of the true

state of the environment. For example, imagine a simple scenario where the agent must learn to lift an object off the ground. Here, a reward useful for task success would be proportional to the height of the object from the ground — a quantity non-trivial to obtain without full state information. Thus, significant effort has been expounded in developing methods that can learn reward functions either explicitly or implicitly from demonstrations, i.e., imitation learning [287, 262, 1, 421]. With these methods, agent policies can either be directly extracted from the demonstrations or trained to optimize rewards functions learned from them.

Imitation learning (IL), however, only somewhat alleviates the need for expert human intervention. First, instead of designing complex reward functions, expert supervision is needed to collect massive datasets such as RT-1 [38], Bridge Dataset [90], D4RL [104], or Robonet [74]. The performance of imitation learning algorithms and their ability to generalize hinges on the coverage and size of data [174, 175], making the collection of large datasets imperative. Second and most importantly, the interface for collecting demonstrations for IL is tedious, requiring expert robot operators to collect thousands of demonstrations. On the contrary, a more intuitive way to define rewards would be in the form of a textual description (e.g., "*robot grasping object*"), or in the form of a naturalistic video demonstration of the task performed by a human actor in an environment separate from the robotic environment. For example, demonstrating to a robot how to open a cabinet door in one's own kitchen is more naturalistic than collecting many thousands of trajectories via teleoperation in the target robotic environment.

Thus, there exists an unmet need for IL algorithms that 1) require very few demonstrations and 2) allow for a natural interface for providing these demonstrations. For instance, algorithms that can effectively learn from language instructions or human demonstrations without the need for full environment state information. Our key insight is that by leveraging Video-and-Language Models (VLMs)—which are already pretrained on large amount of video demonstration and language pairs—we do not need to rely on large-scale and in-domain datasets. Instead, by harnessing the power of VLM embeddings, we treat the mismatch

between a single *instruction's* embedding (provided as a language command or a video demonstration) and the embedding of the video of the current policy's rollout as a proxy reward that will guide the policy towards the desired *instruction*.

To this end, we present RoboCLIP, an imitation learning algorithm that learns and optimizes a reward function based on a single language or video demonstration. The backbone model used in RoboCLIP is S3D [377] trained on the Howto100M dataset [242], which consists of short clips of humans performing activities with textual descriptions of the activities. These videos typically consist of a variety of camera angles, actors, lighting conditions, and backgrounds. We hypothesize that VLMs trained on such diverse videos are invariant to these extraneous factors and generate an actor-agnostic semantically-meaningful representation for a video, allowing them to generalize to unseen robotic environments.

We present an overview of RoboCLIP in Figure 9.1. RoboCLIP computes a similarity score between videos of online agent experience with a task descriptor, i.e., a text description of the task or a single human demonstration video, to generate trajectory-level rewards to train the agent. We evaluate RoboCLIP on the Metaworld Environment suite [393] and on the Franka Kitchen Environment [118], and find that policies obtained by pretraining on the RoboCLIP reward result in $2-3\times$ higher zero-shot task success in comparison to state-of-the-art imitation learning baselines. Additionally, these rewards require no experts for specification and can be generated using naturalistic definitions like natural language task descriptions and human demonstrations.

9.2 Related Work

Learning from Human Feedback. Learning from demonstrations is a long-studied problem that attempts to learn a policy from a dataset of expert demonstrations. Imitation learning (IL) methods, such as those based on behavioral cloning [287], formulate the problem as a supervised learning over state-action

pairs and typically rely on large datasets of expert-collected trajectories directly demonstrating how to perform the target task [38, 219]. However, these large demonstration datasets are often expensive to collect. Another IL strategy is *inverse* RL, i.e., directly learning a reward function from the demonstrations [262, 1, 421, 100]. Inverse RL algorithms are typically difficult to apply when state and action spaces are high-dimensional. Methods such as GAIL [138], AIRL [105], or VICE [106] partially address these issues by assigning rewards which are proportional to the probability of a given state being from the demonstration set or a valid goal state as estimated by a learned discriminator network. However these discriminator networks still require many demonstrations or goal states to train to effectively distinguish between states from agent-collected experience and demonstration or goal states. On the other hand, RoboCLIP's use of pretrained video-and-language models allows us to train agents that learn to perform target tasks with just *one demonstration* in the form of a video or a language description. Other works instead use human feedback in the form of pairwise comparisons or rankings to learn preference reward functions [59, 304, 30, 248, 31, 41, 29, 181, 134]. These preferences may require less human effort to obtain than reward functions, e.g., through querying humans to simply rank recent trajectories. Yet individual trajectory preferences convey little information on their own (less than dense reward functions) and therefore humans need to respond to many preference queries for the agent to learn useful reward functions. In contrast, RoboCLIP is able to extract useful rewards from a single demonstration or single language instruction.

Large Vision and Language Models as Reward Functions. Kwon et al. [177] and Hu and Sadigh [143] propose using large language models (LLMs) for designing and regularizing reward functions that capture human preferences. These works study the reward design problem in text-based games such as negotiations or card games, and thus are not grounded in the physical world. RoboCLIP instead leverages video-and-language models to assess if video demonstrations of robot policies align with an expert demonstration. Prior work has demonstrated that video models can be used as reward functions. For example, Chen, Nair, and Finn [50] learn a visual reward function using human data and then utilize this

reward function for visual model-based control of a robot. However, they require training the reward model on paired human and robot data from the deployment environment. We demonstrate that this paired data assumption can be relaxed by utilizing large-scale vision-language models pretrained on large corpora of human-generated data. The most well-known of these is CLIP [289], which is trained on pairs of images and language descriptions scraped from the internet. While CLIP is trained only on images, video-language-models (VLMs) trained on videos of humans performing daily tasks such as S3D [377] or XCLIP [265] are also widely available. These models utilize language descriptions while training to supervise their visual understanding so that *semantically* similar vision inputs are embedded close together in a shared vector space. A series of recent works demonstrate that these VLMs can produce useful rewards for agent learning. Fan et al. [98] finetune CLIP on YouTube videos of people playing Minecraft and demonstrate that the finetuned CLIP model can be used as a language-conditioned reward function to train an agent. DECKARD [267] then uses the fine-tuned reward function of Fan et al. [98] to reward an agent for completing tasks proposed by a large-language model and abstract world model. PAFF [109] uses a fine-tuned CLIP model to align videos of policy rollouts with a fixed set of language skills and relabel experience with the best-aligned language label. We demonstrate that *videos* and multi-modal task specifications can be utilized to learn reward functions allowing for training agents. Additionally, we present a method to test the alignment of pretrained VLMs with deployment environments.

9.3 Method

Overview. RoboCLIP utilizes pretrained video-and-language models to generate rewards for online RL agents. This is done by providing a sparse reward to the agent at the end of the trajectory which describes the similarity of the agent's behavior to that of the demonstration. We utilize video-and-language models as they provide the flexibility of defining the task in terms of natural language descriptions or video

demonstrations sourced either from the target robotic domain or other more naturalistic domains like human actors demonstrating the target task in their own environment. Thus, a demonstration (textual or video) and the video of an episode of robotic interaction are embedded into the semantically meaningful latent space of S3D [377], a video-and-language model pretrained on diverse videos of human actors performing everyday tasks taken from the HowTo100M dataset [242]. The two vectors are subsequently multiplied using a scalar product generating a similarity score between the 2 vectors. This similarity score (without scaling) is returned to the agent as a reward for the episode.

Notation. We formulate the problem in the manner of a POMDP (Partially Observable Markov Decision Process) with $(\mathcal{O}, \mathcal{S}, \mathcal{A}, \phi, \theta, r, T, \gamma)$ representing an observation space $\mathcal{O}$, state space $\mathcal{S}$, action space $\mathcal{A}$, transition function ϕ, emission function θ, reward function r, time horizon T, and discount factor γ. An agent in state $\boldsymbol{b}s_t$ takes an action $\boldsymbol{b}a_t$ and consequently causes a transition in the environment through $\phi(\boldsymbol{b}s_{t+1} \mid \boldsymbol{b}s_t, \boldsymbol{b}a_t)$. The agent receives the next state $\boldsymbol{b}s_{t+1}$ and reward $r_t = r(\boldsymbol{b}o_t, \boldsymbol{b}a_t)$ calculated using the observation $\boldsymbol{b}o_t$. The goal of the agent is to learn a policy $\boldsymbol{b}\pi$ which maximizes the expected discounted sum of rewards, i.e., $\sum_{t=0}^{T} \gamma^t r_t$. Note that all of our baselines utilize the true state for reward generation and for policy learning. To examine the effect of using a video-based reward, we also operate our policy on the state space while using the pixel observations for reward generation. Thus, r_t uses $\boldsymbol{b}o_t$ while $\boldsymbol{b}\pi$ uses $\boldsymbol{b}s_t$ for RoboCLIP while for all other baselines, both r_t and $\boldsymbol{b}\pi$ utilize $\boldsymbol{b}s_t$. This of course is unfair to our method, but we find that in spite of the advantage provided to the baselines, RoboCLIP rewards still generate higher zero-shot success.

Reward Generation. During the pretraining phase, we supply the RoboCLIP reward to the agent in a sparse manner at the end of each episode. This is done by storing the video of an episode of the interaction of the agent with the environment into a buffer as seen in Figure 9.1. A sequence of observations of length 128 are saved in a buffer corresponding to the length of the episode. S3D is trained on videos length 32

frames and therefore the episode video is subsequently downsampled to result in a video of length $T = 32$. The video is subsequently center-cropped to result in frames of size $(250, 250)$. This is done to ensure that the episode video is preprocessed to match the specifications of the HowTo100M preprocessing used to train the S3D model. Thus the tensor of a sequence of T observations $\boldsymbol{b}o_{0:T}$ is encoded as the latent video vector $\boldsymbol{b}z^v$ using

$$\boldsymbol{b}z^v = S3D^{\text{video-encoder}}(\boldsymbol{b}o_{0:T}) \tag{9.1}$$

The task specification is also encoded into the same space. If it is defined using natural language, the language encoder in S3D encodes a sequence of K textual tokens $\boldsymbol{b}d_{0:K}$ into the latent space using:

$$\boldsymbol{b}z^d = S3D^{\text{text-encoder}}(\boldsymbol{b}d_{0:K}) \tag{9.2}$$

If the task description is in the form of a video of length K, then we preprocess and encode it using the video-encoder in S3D just as in Equation (9.1). For intermediate timesteps, i.e., timesteps other than the final one in an episode, the reward supplied to the agent is zero. Subsequently, at the end of the episode, the similarity score between the encoded task descriptor $\boldsymbol{b}z^d$ and the encoded video of the episode $\boldsymbol{b}z^v$ is used as reward $r^{\text{RoboCLIP}}(T)$. Thus the reward is:

$$r^{\text{RoboCLIP}}(t) = \begin{cases} 0, & t \neq T \\ \boldsymbol{b}z^d \cdot \boldsymbol{b}z^v & t = T \end{cases}$$

where $\boldsymbol{b}z^d \cdot \boldsymbol{b}z^v$ corresponds to the scalar product between vectors $\boldsymbol{b}z^d$ and $\boldsymbol{b}z^v$.

Agent Training. Using r^{RoboCLIP} defined above, we then train an agent online in the deployment environment with any standard reinforcement learning (RL) algorithm by labeling each agent experience trajectory with r^{RoboCLIP} after the agent collects it. In our paper, we train with PPO [311], an on-policy

RL algorithm, however, RoboCLIP can also be applied to off-policy algorithms. After training with this reward, the agent can be zero-shot evaluated or fine-tuned on true environment reward on the target task in the deployment environment.

9.4 Experiments

We test out each of the hypotheses defined in Section 9.1 on simulated robotic environments. Specifically, we ask the following questions:

1. *Do existing pretrained VLMs semantically align with robotic manipulation environments?*
2. *Can we utilize natural language to generate reward functions?*
3. *Can we use videos of expert demonstrations to generate reward functions?*
4. *Can we use out-of-domain videos to generate reward functions?*
5. *Can we generate rewards using a combination of demonstration and natural language?*
6. *What aspects of our method are crucial for success?*

We arrange this section to answer each of these questions. Both RoboCLIP and baselines utilize PPO [311] for policy learning.

Baselines. We use 2 state-of-the-art methods in inverse reinforcement learning: **GAIL**, or Generative Adversarial Imitation Learning [138] and **AIRL** or Adversarial Inverse Reinforcement Learning [105]. Both of these methods attempt to learn reward functions from demonstrations provided to the agent. Subsequently, they train an agent using this learned reward function to imitate the expert behavior. Both methods receive a single demonstration, consistent with our approach of using a single video imitation. However, since they both operate on the ground-truth environment state, we provide them with a **trajectory of states**, instead of images, thereby providing them privileged state information that our method does not receive.

9.4.1 Domain Alignment

Pretrained vision models are often trained on a variety of human-centric activity data, such as Ego4D [112]. Since we are interested in solving robotic tasks with view from third person perspectives, we utilize the S3D [377] VLM pretrained on HowTo100M [242], a dataset of short third-person clips of humans performing everyday activities. This dataset, however, contains no robotic manipulation data.

To analyze the alignment of the VLM to different domains, we perform a confusion matrix analysis using videos from Metaworld [393]. We collect 10 videos per task with varying values of true reward. For each video, we also collect the true reward. We then compute the RoboCLIP reward for each video using VLM alignment between the textual description of the task and the video. We visualize the correlations between the RoboCLIP and true rewards in the form of an $n \times n$ matrix where entry (i, j) corresponds to the correlation between the true reward and the RoboCLIP reward generated for the i^{th} task using the j^{th} text description. As one can see, for a given task, the highest correlation in the matrix is for the correct textual description. We visualize one such similarity matrix in Figure 9.2 for Metaworld. We find that Metaworld seems to align well in the latent space of the model with a more diagonal-heavy confusion matrix. The objects are all correctly identified.

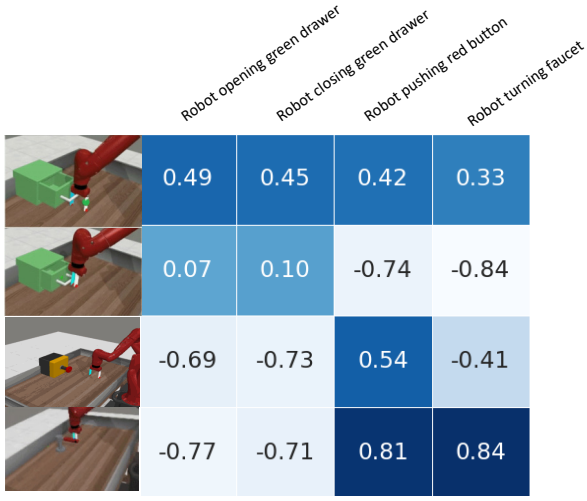


Figure 9.2: **RoboCLIP: Domain Alignment.** We perform a confusion matrix analysis on a subset of the data on collected on Metaworld [393] environments by comparing the pair-wise similarities between the latent vectors of the strings describing the videos and those of the videos. We find that Metaworld is well-aligned with higher scores along the diagonal than along the off-diagonal elements.

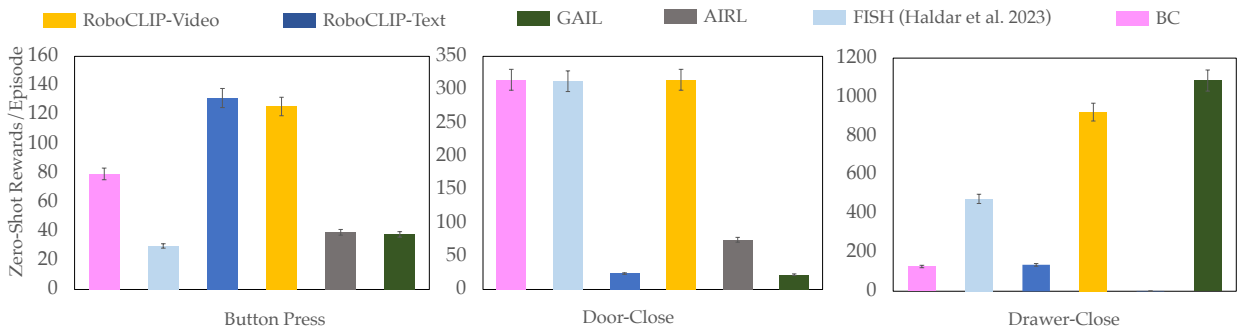


Figure 9.3: **RoboCLIP: Language Rewards.** The pretrained VLM is used to generate rewards via the similarity score of the encoding of an episode of interaction of an agent in its environment, $\boldsymbol{b}z^v$ with the encoding of a task specifier $\boldsymbol{b}z^d$ specified in natural language. We use the strings, "*robot closing black box*", "*robot closing green drawer*" and "*robot pushing red button*" for conditioning for the 3 environments respectively. We find that agents pretrained on these language-conditioned rewards outperform imitation learning baselines like GAIL [138] and AIRL [105].

9.4.2 Language for Reward Generation

The most naturalistic way to define a task is through natural language. We do this by generating a sparse reward signal for the agent as described in Section 9.3: the reward for an episode is the similarity score between its encoded video and the encoded textual description of the expected behavior in the VLM's latent space. The reward is provided to the agent at the end of the episode. For RoboCLIP, GAIL, and AIRL, we first pretrain the agents online with their respective reward functions and then perform finetuning with the true task reward in the deployment environment. We perform this analysis on 3 Metaworld Environments: `Drawer-Close`, `Door-Close` and `Button-Press`. We use the textual descriptions, "*robot closing green drawer*", "*robot closing black box*", and "*robot pushing red button*" for each environment, respectively. Figure 9.3 plots returns on the target tasks while finetuning on the depoloyment environment after pretraining (with the exception of the Dense Task Reward baseline). Our method outperforms the imitation learning baselines with online exploration in terms of true task rewards in all environments. Additionally our baselines utilize the full state information in the environment for reward generation where RoboCLIP

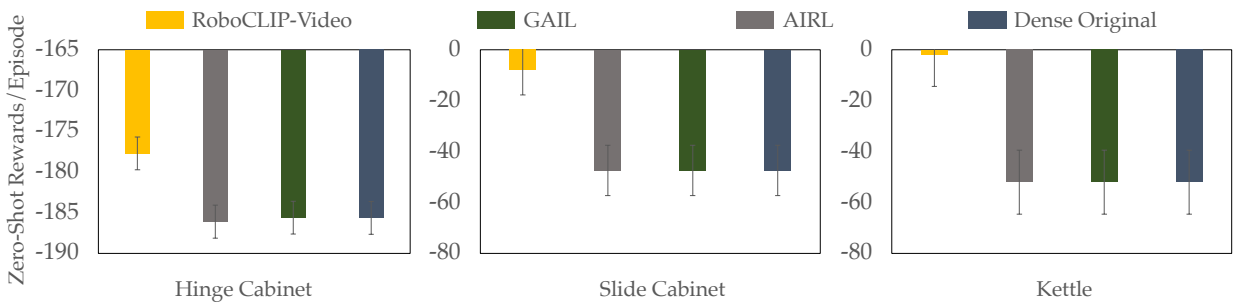


Figure 9.4: **RoboCLIP: In-Domain Videos.** The pretrained VLM is used to generate rewards via the similarity score of the encoding of an episode of interaction of an agent in its environment, $\boldsymbol{b}z^{v}$ with the encoding of a video demonstration of expert behavior in the same environment. The similarity score between the latent vectors is provided as reward to the agent and is used to train online RL methods. We study this setup in the `Kettle`, `Hinge` and `Slide` Tasks in the Franka Kitchen Environment [118]. We find that policies trained on the RoboCLIP reward are able to learn to complete the task in all three setups without any need for external rewards using just a single in-domain demonstration.

uses only the pixels to infer state. RoboCLIP also achieves more than double zero-shot rewards in all environments — importantly, the RoboCLIP-trained agent is able to complete the tasks even before finetuning on true task rewards.

9.4.3 In-Domain Videos for Reward Generation

Being able to use textual task descriptors for reward generation can only work in environments where there is domain alignment between the pretrained model and the visual appearance of the environment. Additionally, VLMs are large models often with billions of parameters making it computationally expensive to fine tune for domain alignment. The most naturalistic way to define a task in such a setting is in the form a single demonstration in the robotic environment which can be collected using teleoperation. We study how well this works in the Franka Kitchen [118] environment. We consider access to a single demonstration per task whose video is used to generate rewards for online RL.

Quantitative Results. We measure the zero-shot task reward, which increases as the task object (i.e., Kettle, Slide and Hinge Cabinets) gets closer to its goal position. This reward does not depend on the position of the end-effector, making the tasks difficult. Figure 9.4 shows the baselines perform poorly as

they generally do not interact with the target objects, while RoboCLIP is able to solve the task using the reward generated using the video of a single demonstration.

Qualitative Results. We find that RoboCLIP allows for mimicking the "style" of the source demonstration, with idiosyncrasies of motion from the source demonstration generally transferring to the policy generated. We find this to occur in the kitchen environment's `Slide` and `Hinge` task as seen in Figure 9.5. The first row of the subfigures in Figure 9.5 are visualizations of the demonstration video used to condition the VLM for reward generation. The bottom rows correspond to the policies that are trained with the generated rewards of RoboCLIP. As can be seen, the `Slide` demonstration consists of a wide circular arc of motion. This is mimicked in the learned policy, although the agent misses the cabinet in the first swipe and readjusts to make contact with the handle.

This effect is even more pronounced in the `Hinge` example where the source demonstration consists of twirling wrist-rotational behavior, which is subsequently imitated by the learned policy. The downstream policy misses the point of contact with the handle but instead uses the twirling motion to open the hinged cabinet in an unorthodox manner by pushing near the hinge. We posit that the VLMs used in RoboCLIP contain a rich latent space encoding these various motions, and so even if they cannot contain semantically meaningful latent vectors in the Franka Kitchen environments due to domain mismatch, they are still able to encode motion information allowing them to be used for RoboCLIP with a single demonstration video.

9.4.4 Out-of-Domain Videos for Reward Generation

Another natural way to define a task is to demonstrate it yourself. To this end, we try to use demonstrations of humans or animated characters acting in separate environments as task specification.

For this, we utilize animated videos of a hand pushing a red button and opening a green drawer and a real human video of opening a fridge door (see Figure 9.7). The animated videos are collected from stock image repositories and the human video is collected using a phone camera in our lab kitchen. Using the

Figure 9.5: **RoboCLIP: Imitation Analysis.** The first row in each subfigure shows the visualizations of the demonstration video used for reward generation via the VLM. The second rows are videos taken from policy recovered from training on the RoboCLIP reward generated using the videos in the first rows. The quick swiping motion demonstrated in the `Slide` demonstration is mimicked well in the resultant policy while the wrist-rotational "trick-shot" behavior in the demonstration for `Hinge` appears in the resultant learned policy.

encodings of these video, we test out RoboCLIP in the 3 corresponding Metaworld tasks - `Button-Press`, `Drawer-Open` and `Door-Open`. We follow the same setup as in Section 9.4.2 by first pretraining methods with their respective reward functions and then finetuning in the deployment environment with target task reward.

We compare the performance of the policy trained with these rewards to GAIL [138] and AIRL [105] trained using the same single expert demonstration as RoboCLIP on these rewards with state information. These methods are known to be data-hungry, requiring multiple demonstrations to train their reward functions. Consequently, they perform much worse than RoboCLIP, even with 2-3x worse zero-shot task performance, as can be seen from Figure 9.7.

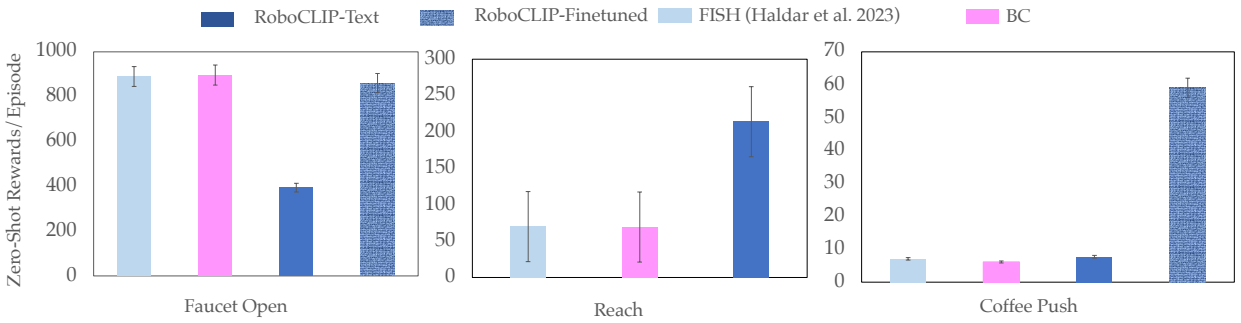


Figure 9.6: **RoboCLIP: Finetuning Results.** In harder environments, like `Coffee-Push` and `Faucet-Open`, we find that RoboCLIP rewards do not solve the task completely. We test whether providing a single demonstration in the environment (using states and actions) is enough to finetune this pretrained policy, a setup identical to our baselines. Thus, we pre-train on the RoboCLIP reward from language and then finetune using a single robotic demonstration. This improves performance by $\sim$ 200%. See videos on our website.

9.4.5 Multimodal Task Specification

Using videos to specify a task description is possible when either there is access to a robot for teleoperation as in Section 9.4.3 or a human can demonstrate a behavior in their own environment as in Section 9.4.4. When these are not the case, a viable alternative is to utilize multimodal demonstrations. For example, consider a scenario where the required task is to push a drawer to close it, but only a demonstration for pushing a button is available. In this situation, being able to edit the video of the off-task demonstration is useful. This way, one can direct the agent to move its end-effectors to push the drawer instead of the button.

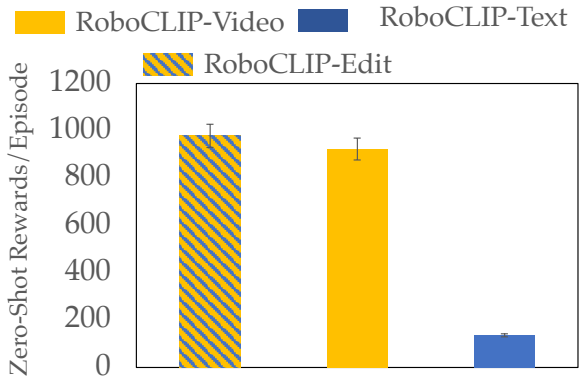


Figure 9.8: **RoboCLIP: Multimodal Tasks.** We study whether video demonstrations of expert demonstrations can be used to define tasks. We use the latent embedding of a video demonstration of a robot pushing a button and subtract from it the embedding of the text "*red button*" and add to it the embedding of the text "*green drawer*". This modified latent is used to generate rewards in the Drawer-Close environment. We find that the policy trained using this modified vector outperforms string-only manipulation in the zero-shot setting.

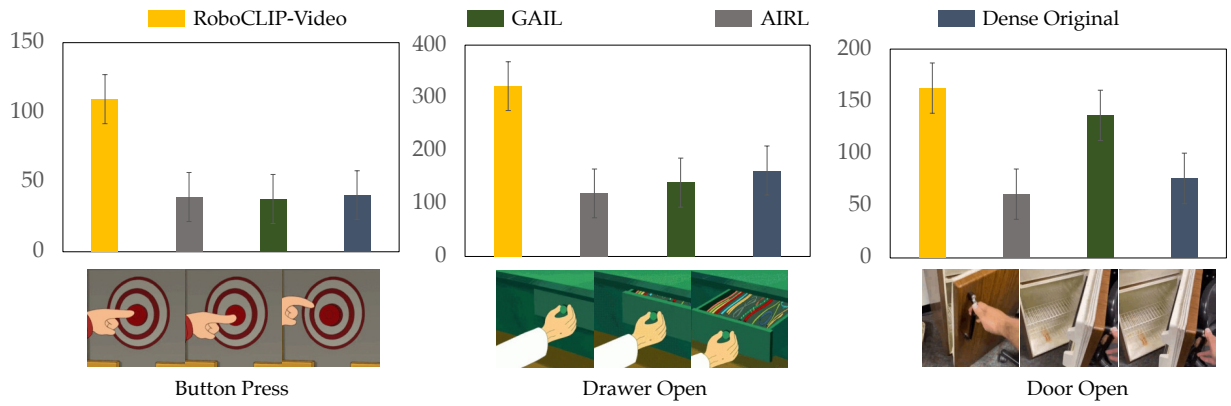


Figure 9.7: **RoboCLIP: Out-of-Domain Videos.** A Pretrained Video-and-Language Model is used to generate rewards via the similarity score of the encoding of an episode of interaction of an agent in its environment, $\boldsymbol{bz}^{v}$ with the encoding of a task specifier $\boldsymbol{bz}^{d}$ in the form of a video of a human or an animated character demonstrating a task in their own environment. The similarity score between the latent vectors is provided as reward to the agent and is used to train online RL methods. The frames below the graphs illustrate the video used for reward generation.

We do this by algebraically modifying the encoding of the video demonstration:

$$\boldsymbol{bz}^{\text{edited}}(\text{push drawer}) = \boldsymbol{bz}^{video}(\text{push button}) - \boldsymbol{bz}^{text}(\text{button}) + \boldsymbol{bz}^{text}(\text{drawer}) \tag{9.3}$$

where $\boldsymbol{bz}^{\text{edited}}(\text{push drawer})$ is the vector used to generate rewards in the `Drawer-Close` environment, $\boldsymbol{bz}^{video}(\text{push button})$ is the vector of the encoding of the video of the robot pushing a button, $\boldsymbol{bz}^{text}(\text{button})$ is the encoding of the string *button* and $\boldsymbol{bz}^{text}(\text{drawer})$ is the encoding of the string *drawer*. As can be seen in Figure 9.8, defining rewards in such a multimodal manner results in a higher zero-shot score than the dense task reward and also pretraining on the string-only task reward.

9.4.6 Finetuning

In harder environments, and with rewards from OOD videos and language, the robot policy sometimes approaches the target object, but fails to complete the task. Thus, we tested whether providing a single demonstration (using states and actions) was enough to finetune this pretrained policy.

Thus, for this experiment we first (1) pretrain on the RoboCLIP reward from human videos or language descriptions and then (2) finetune using a single demonstration. As seen in Figure 9.6, we find that this converts each of the partially successful policies into complete success and improves the rewards attained by the policies by 200%. This fine-tuning setup is especially useful in harder tasks like like `Coffee-Push` and `Faucet-Open` and is competitive with state-of-the-art approaches like FISH [126].

9.4.7 Ablations

Finally, we investigate the effects of various design decisions in RoboCLIP. First, we study the effect of additional video demonstrations on agent performance. We also examine the necessity of using a pre-trained VLM. Recent works like RE3 [313, 357] have shown that randomly initialized networks often contain useful image priors and can be used to supply rewards to agents to encourage exploration. Therefore, we test whether a *randomly initialized* S3D VLM can supply useful pretraining rewards in the in-domain video demonstration setup as in Section 9.4.3. Finally, we study our choice of pretrained VLM. We examine whether a pretrained CLIP [289], which encodes single images instead of videos and was trained on a different dataset from S3D, can be used to generate rewards for task completion. In this setup, we record the last image in an episode of interaction of the agent in its environment and feed

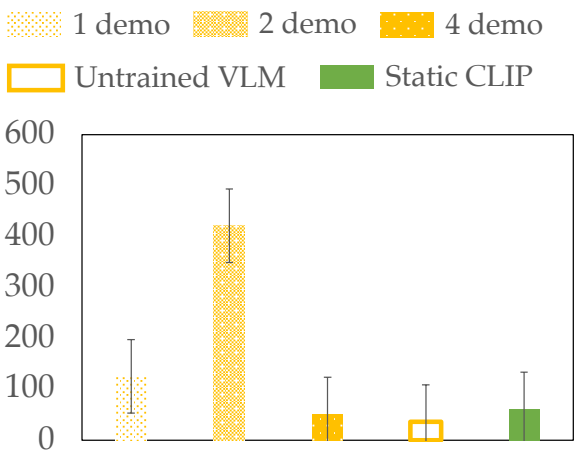


Figure 9.9: **RoboCLIP: Ablation Studies.** We study the effects of varying the number of demonstrations provided to the agent can have on downstream rewards. We also study the effects of the training provided to the VLM on the downstream rewards. Finally, we study whether using CLIP trained on static images provides good rewards for pretraining.

it to CLIP trained on ImageNet [303] (i.e., not trained on videos). We then specify the task in natural language and use the similarity between the embeddings of the textual description of the task and the final image in the episode to generate a reward that is fed to the agent for online RL.

As seen in Figure 9.9, using a single video demonstration provides the best signal for pretraining. We posit that our method performs worse when conditioned on multiple demonstrations as the linear blending of multiple video embeddings, which is used due to the scalar product, does not necessarily correspond to the embedding of a successful trajectory. Crucially, we also find that using the static image version of CLIP does not provide any useful signal for pretraining. The zero-shot performance is very poor, which we posit is because it does not contain any information about the dynamics of motion and task completion although it contains semantic meaning about objects in the frame. On the other hand, video contrastive learning approaches do contain this information. This is further evidenced by the fact that inspite of poor domain alignment between Franka Kitchen and the VLM, we find that encodings of in-domain video demonstrations are still good for providing a pretraining reward signal to the agent.

9.5 Conclusion

Summary. We studied how to distill knowledge contained in large pretrained Video-and-Language-Models into online RL agents by using them to generate rewards. We showed that our method, RoboCLIP, can train robot policies using a single video demonstration or textual description of the task, depending on how well the domain aligns with the VLM. We further investigated alternative ways to use RoboCLIP, such as using out-of-domain videos or multimodal demonstrations. Our results showed RoboCLIP outperforms the baselines in various robotic environments.

Limitations and Broader Impact. Since we are using VLMs, the implicit biases within these large models could percolate into RL agents. Addressing such challenges is necessary, especially since it is unclear what the form of biases in RL agents might look like. Currently, our method also faces the challenge of

stable finetuning. We find that in some situations, finetuning on downstream task reward results in instabilities as seen in the language conditioned reward curve in Figure 9.8. This instability is potentially due to the scale of rewards provided to the agent. Rewards from the VLM are fairly low in absolute value and subsequently, the normalized Q-values in PPO policies are out-of-shape when finetuned on task rewards. In our experiments, this is not a big problem since the RoboCLIP reward is already sufficient to produce policies that complete tasks without any deployment environment finetuning, but this will be essential to solve when deploying this for longer horizon tasks.

Another limitation of our work is that there is no fixed length of pretraining. Our current method involves pretraining for a fixed number of steps and then picking the best model according to the true task reward. This is of course difficult when deploying RoboCLIP in a real-world setup as a true reward function is unavailable and a human must monitor the progress of the agent. We leave this for future work.

Chapter 10

ReWiND: Language-Guided Rewards Teach Robot Policies without New Demonstrations

Figure 10.1: **Overview.** We pre-train a policy and reward model from a small set of language-labeled demos. Then, we solve unseen task variations via language-guided RL—*without additional demos.*

10.1 Introduction

A great teacher does not just tell you if you are right or wrong. Instead, they guide you by providing feedback when you make mistakes, highlighting progress as you learn something new, and adapting to how you learn best. For deployed robots to learn new tasks in the wild, they need similarly intelligent teachers. These teachers—in the form of robust reward models—should: (1) offer *dense, informative feedback*, especially during failures; (2) *generalize* their guidance to unseen tasks; and (3) remain *robust* to diverse robot behaviors during its learning process. Our paper leverages these insights to develop reward models capable of teaching robots unseen tasks.

In this work, we introduce ReWiND (**Re**wards **Wi**thout **N**ew **D**emonstrations), a framework designed to teach robots unseen tasks in a sample-efficient manner using only a few grounding human demonstrations for training tasks (see Figure 10.1). Typically, teaching robots involves large-scale imitation learning [39, 37, 32, 192], where human experts provide demonstrations for each new task. However, collecting task-specific demonstrations is expensive and time-consuming. Reinforcement learning (RL) offers a more autonomous alternative by using reward functions as teachers, allowing robots to learn through interaction. Yet, manually designing these reward functions demands substantial manual effort and domain-specific expertise [341]. Recent progress in language-conditioned reward learning [334, 177, 143, 396, 223, 224, 195, 10, 368, 388] has aimed at addressing these challenges, but often assumes unrealistic conditions such as availability of ground-truth states [177, 143, 396, 223, 224, 195], thousands of demonstrations [10], or online training of reward models from scratch [368, 388], limiting their practical applicability.

ReWiND overcomes these challenges by instead assuming only a handful of demonstrations—e.g., five per task—to enable real-world robot learning of unseen task variations. ReWiND first trains a language-conditioned reward model from these demonstrations, then uses it to pre-train a language-conditioned policy via offline RL. When deployed, ReWiND efficiently fine-tunes the policy on new task variations by reward-labeling *online* interaction episodes.

Our core contribution is in designing ReWiND's reward model to capture three key properties outlined earlier: **dense feedback**, **generalization**, and **robustness**. First, to provide *dense, informative feedback*, we design a *cross-modal sequential aggregator* that leverages pre-trained vision and language embeddings to predict *progress* within demonstration videos. Progress prediction offers a stable, densely supervised training signal that naturally translates into a dense reward function. We also introduce *video rewinding* to generate failure trajectories from successful demonstrations, allowing ReWiND to provide dense reward feedback even when the policy is making mistakes. Then, to ensure *generalization* across unseen tasks and *robustness* to diverse behaviors, we incorporate targeted inductive biases into the cross-modal sequential

aggregator architecture and supplement training with diverse robotics data from Open-X [67], enabling the reward model to extrapolate to novel visual and linguistic scenarios.

We introduce reward metrics measuring the above properties on which ReWiND achieves **23-74%** relative improvements over reward learning baselines. Further, comprehensive success rate evaluations on Metaworld manipulation tasks and a real-world bimanual robot setup demonstrate ReWiND beats baselines by **2X** in simulation and improves real-world pre-trained policies by **5X**.

10.2 Related Works

Learning Reward Functions. Prior work in reinforcement learning has proposed various methods for *learning* reward functions. Examples include inverse RL [262, 1, 421, 100], where reward functions are learned from demonstrations, or methods where rewards are implicitly learned from expert or goal state distributions [138, 105, 106]. However, these works require new target-task demonstrations to reward unseen tasks. ReWiND instead trains a general, language-conditioned reward function from an initial demonstration set to reward unseen task variations without further demos.

Another line of work learns reward functions directly from human feedback in the form of comparisons [60, 304, 29, 181, 134], reward sketches [45], preference rankings [248], scaled preferences [372], critiques [69], corrections [22], interventions [170], and language [388]. While these feedback types may require less human effort than demonstrations or manually written reward functions, these works still require humans to provide extensive feedback for each unseen task.

Reward Generation with Pre-trained Models. Prior work has also explored using large pre-trained models to generate reward functions instead of learning them from scratch. Some approaches use LLMs

to generate language-conditioned rewards [177, 143, 396, 223, 224, 195], but they typically rely on ground-truth state information that is difficult to obtain in real-world settings. In contrast, ReWiND generates rewards from just a task description and a policy execution video.

Other approaches use pre-trained vision models to derive rewards from visual observations [51, 71, 98, 267, 255, 334, 368, 10, 263, 386, 387, 220, 300, 264, 164]. Among these, RoboCLIP [334], LIV [223], VLC [10], and GVL [224]—like ReWiND—reward unseen robot manipulation tasks directly from language without additional target-task demos or online tuning. We show in Section 10.4.1 that these baselines underperform ReWiND in rewarding policies in our limited-data setting. Most similar to ReWiND, Foundation Actor-Critic (FAC) [390] enables efficient RL from language via potential-based shaping rewards from a pre-trained VLM. However, FAC depends on predefined policy priors (e.g., code-based primitives from LLMs), whereas ReWiND learns them through offline RL on non-target tasks.

10.3 ReWiND: Learning Rewards Without New Demonstrations

We study the problem of learning unseen, language-specified tasks in a *target environment*, formulated as a Markov decision process (MDP). The *target environment* refers to the deployment scene (e.g., a robot tabletop). We train a policy $\pi_\theta(a_t \mid o_t, z)$ that selects actions a_t based on images o_t and language instructions z.* The policy is optimized to maximize rewards predicted by a learned reward function $R_\psi(o_{1:t}, z)$, which conditions on the frame sequence $o_{1:t}$ and instruction z to output per-timestep estimated rewards $\hat{r}_t$. We assume access to a small demonstration dataset $\mathcal{D}_{\text{demos}}$ in the target environment containing 15–20 tasks with $\sim$5 demonstrations each. Following prior definitions of generalization [17, 108], we define a task as unseen if it requires a novel *action sequence*, its distribution of *image observations* has changed, or needs a new *language instruction*.

*Proprioception (e.g., end-effector positions) can also be included but is omitted here for simplicity.

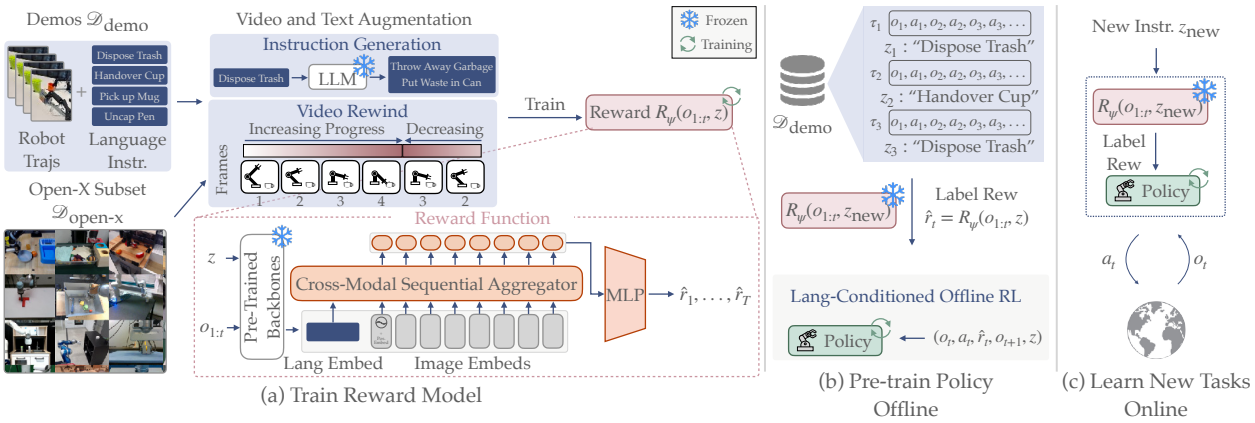


Figure 10.2: **(a):** We train a reward model $R_\psi(o_{1:t}, z)$ on a small demonstration dataset $\mathcal{D}_{\text{demos}}$ and a curated subset of Open-X, $\mathcal{D}_{\text{open-x}}$, augmented with LLM-generated instructions and video rewinding. $R_\psi(o_{1:t}, z)$ predicts video *progress* rewards $\hat{r}_{1:T}$ from pre-trained embeddings of image observations $o_{1:T}$ and language instructions z, and assigns 0 progress to misaligned video-language pairs. **(b):** We use the trained $R_\psi(o_{1:t}, z)$ to label $\mathcal{D}_{\text{demos}}$ with rewards and pre-train a language-conditioned policy using offline RL. **(c):** For an unseen task specified by z_{new}, we fine-tune π with online rollouts and reward labels from $R_\psi(o_{1:t}, z_{\text{new}})$.

ReWiND consists of 3 phases (see Figure 10.2): (1) learning a reward function from limited target environment demos, then (2) pre-training π with learned rewards on the demos, and finally (3) using the reward function and pre-trained policy to learn a new language-specified task online.

10.3.1 Learning a Reward Function

Our primary objective for reward prediction is regressing directly to per-frame *progress* within an observation sequence $o_{1:T}$ conditioned on instruction z. Unlike prior methods using relative targets [386, 10], our progress-based objective provides fixed targets that are more stable to train on, and translates directly into a dense, $[0, 1]$-normalized reward for policy training. To ensure robustness against mismatched observations and instructions, we also sample unrelated observation sequences $o^{\text{other}}_{1:T}$ and train $R_\psi(o_{1:t}, z)$ to predict zero progress. Our reward prediction loss is:

$$\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T}) = \sum_{t=1}^{T}(R_\psi(o_{1:t}, z) - \underbrace{t/T}_{\text{matched seq. progress}})^2 + \sum_{t=1}^{T} \underbrace{R_\psi(o^{\text{other}}_{1:t}, z)^2}_{\text{mismatched seq. 0 progress}} . \tag{10.1}$$

However, simply training a neural network $R_\psi(o_{1:t}, z)$ on $\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T})$ with a small set of demonstrations is unlikely to ensure that it can train a policy on unseen tasks. $R_\psi(o_{1:t}, z)$ should:

D1 **Generalize to new tasks**, i.e., new policy execution videos and instructions not in $\mathcal{D}_{\text{demos}}$.

D2 **Produce rewards aligned with *policy rollouts***, not just successful demonstration videos.

D3 **Be robust to input variations**, i.e., different ways to solve or specify the task.

To this end, we incorporate diverse off-the-shelf data curated from the Open-X dataset [67] to help with generalization (D1) and robustness (D3), perform targeted video and language augmentations for better reward prediction and language input robustness (D2, D3), and make targeted network architecture choices for generalization (D1). For a visual overview, see Figure 10.2a.

10.3.1.1 Incorporating Diverse Data (D1, D3)

To help $R_\psi(o_{1:t}, z)$ generalize to tasks unseen in $\mathcal{D}_{\text{demos}}$ (D1) and make it robust to diverse ways of executing and specifying tasks (D3), we subsample the Open-X Dataset [67], denoted $\mathcal{D}_{\text{open-x}}$. We specifically select Open-X trajectories with object-centric language instructions, e.g., "*pick coke can from fridge*," or directional instructions, e.g., "*drag the circle to the left of the star*," to help $R_\psi(o_{1:t}, z)$ generalize to objects and directions not contained in $\mathcal{D}_{\text{demos}}$. This dataset contains ∼356k trajectories with ∼59k unique task strings. For detailed dataset information, see **??**.

10.3.1.2 Video and Language Augmentation (D2, D3)

Given our datasets $\mathcal{D}_{\text{demos}}$ and $\mathcal{D}_{\text{open-x}}$, we perform both video and language augmentations that help the reward function accurately predict rewards for unsuccessful *policy execution* videos (D2) and be robust to varied ways of specifying the task instructions z (D3). We call the video augmentation **video rewind** and our text augmentation **instruction generation**.

Video Rewind. Both $\mathcal{D}_{\text{demos}}$ and $\mathcal{D}_{\text{open-x}}$ contain human demonstrations, which are assumed to be successful and of high-quality. Training $R_\psi(o_{1:t}, z)$ on $\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T})$ only using these successful demonstrations, may result in $R_\psi(o_{1:t}, z)$ overfitting to these successful trajectories. However, during online deployment, $R_\psi(o_{1:t}, z)$ will likely encounter failure trajectories (unseen during training) which such an overfit model may reward highly. This is undesirable and prior works attempt to address this issue by explicitly training their reward model on failed trajectories [10], but these trajectories add a great additional burden on demonstrators to collect and must be added post-hoc to any existing dataset, making it harder to scale.

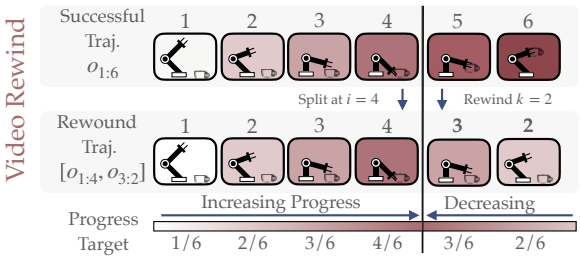


Figure 10.3: **Video rewind.** We split a demo at intermediate timestep i into forward/reverse sections. Here, the forward section shows the robot approaching the cup; the reverse section $(o_{i-1}, o_{i-2}, \ldots)$ resembles dropping it.

Instead, we address this problem in a scalable manner by randomly *rewinding videos*. Consider a video of a robot picking up a cup. If we rewind the video for a few frames right when the robot grabs the cup, it now looks like one in which the robot attempted to grasp the cup and then dropped it.[†] By training $R_\psi(o_{1:t}, z)$ to predict rewards corresponding to *reverse progress* on the rewound subsequence, it (1) is trained on observation sequences mimicking failed policy rollouts that will occur during online RL, and (2) learns to *decrease* reward when necessary. Thus rewinding helps $R_\psi(o_{1:t}, z)$ reward a *policy*—not a human demonstrator—which will help with online RL (**D2**). See Figure 10.3 for a visual example. Formally, rewinding means sampling a random split point i within an observation sequence $o_1 \ldots o_T$, rewinding k (k is also

[†]Random rewinding may result in some physically implausible sequences. However, since they won't appear during inference, the rewards produced by $R_\psi(o_{1:t}, z)$ for such sequences should not affect online RL.

sampled) frames, then appending those k frames to the end of the sequence to become $o_1 \ldots o_i, o_{i-1}, \ldots, o_{i-k}$. The remaining frames from $i+1$ to T are then unused. Our video rewind training objective follows:

$$\mathcal{L}_{\text{rewind}}(o_{1:T}, z) = \sum_{t=1}^{i} \underbrace{(R_\psi(o_{1:t}, z) - \frac{t}{T})^2}_{\text{Loss for original trajectory until } i} + \sum_{t=1}^{k} \underbrace{(R_\psi([o_{1:i}, o_{i-1:i-t}], z) - \frac{i-t}{T})^2}_{\text{Rewound video for } k \text{ frames from } i-1}. \tag{10.2}$$

Instruction Generation. We also generate 5-10 additional language instructions for each task in $\mathcal{D}_{\text{demos}}$ by prompting an LLM. This augmentation helps $R_\psi(o_{1:t}, z)$ with input robustness to possible new task instructions (D3). While training $R_\psi(o_{1:t}, z)$, any time we sample an observation sequence $o_{1:T}$, its instruction z is uniformly randomly sampled from all available matching instructions, generated or original. We did not augment $\mathcal{D}_{\text{open-x}}$ due to its instruction diversity.

10.3.1.3 Architecture (D1)

Due to the limited size of $\mathcal{D}_{\text{demos}}$, we carefully design the architecture for $R_\psi(o_{1:t}, z)$ to maximize generalization to new tasks (D1) while retaining the ability to optimize $\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T})$ well.

Frozen Input Encoders. We use frozen image and language encoders as the backbone of $R_\psi(o_{1:t}, z)$: we use DINOv2 [269] for image encoding due to its strong object-centric representations and `all-MiniLM-L12-v2` [298] for instruction encoding due to its small embedding size ($= 384$). In $R_\psi(o_{1:t}, z)$, we first encode images and instructions: $o^{\text{embed}}_{1:t} = \text{DINO}(o_{1:t})$, $z^{\text{embed}} = \text{MiniLM}(z)$. Then, we train a small *cross-modal sequential aggregator* transformer conditioned on $(o^{\text{embed}}_{1:t}, z^{\text{embed}})$ that learns to aggregate frozen language and image embeddings to generate progress rewards $\hat{r}_t$ directly (see Figure 10.2(a) in the "Reward Function" box).

Positional Embeddings. Finally, the cross-modal sequential aggregator's transformer requires positional information about the frames to properly predict rewards (e.g., for distinguishing "pull" vs. "push").

However, if we naïvely add positional embeddings to each image, it can "cheat" by predicting progress using the positional embeddings. Therefore, similar to how Ma et al. [220] prompt an LLM with the position of the first video frame, we add a positional embedding to the *first* image.

Reward Model Summary. In summary, ReWiND trains a reward function $R_\psi(o_{1:t}, z)$ to predict task progress, using data augmentation (video rewinding, instruction generation) and additional Open-X data ($\mathcal{D}_{\text{open-x}}$) to improve generalization. $R_\psi(o_{1:t}, z)$ combines pretrained vision and language encoders with a lightweight cross-modal sequential aggregator that uses only first-frame positional embeddings. For full implementation details, see Appendix G.1.1. The final objective is:

$$\min_\psi \mathbb{E}_{(o_{1:T}, z, o^{\text{other}}_{1:T}) \sim \mathcal{D}_{\text{demos}}, \mathcal{D}_{\text{open-x}}} \left[\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T}) + \mathcal{L}_{\text{rewind}}(o_{1:T}, z) \right]. \tag{10.3}$$

10.3.2 Policy Learning

Pre-training. After training $R_\psi(o_{1:t}, z)$, we pre-train $\pi_\theta(a_t \mid o_t, z)$ on demonstrations $\mathcal{D}_{\text{demos}}$ labeled with rewards. This pre-training guides $\pi_\theta(a_t \mid o_t, z)$ toward reasonable behaviors during exploration, even if downstream tasks differ from those in $\mathcal{D}_{\text{demos}}$. Given a trajectory with instruction z, $\{(o_t, a_t)\}_1^T$, we assign rewards $\hat{r}_t = R_\psi(o_{1:t}, z)$ at each timestep and add a success bonus to the final reward to encourage reaching the goal despite possibly noisy reward signals:

$$\hat{r}^{\text{off}}_t = R_\psi(o_{1:t}, z) + r_{\text{success}} \cdot \mathbb{1}[t = T]. \tag{10.4}$$

We then train $\pi_\theta(a_t \mid o_t, z)$ via offline RL using tuples $(o_t, a_t, \hat{r}_t, o_{t+1}, z)$. We use IQL [171] as prior work has demonstrated it works on real robots [360, 407, 405]. See Figure 10.2(b) for an overview.

Learning Online. To learn a new task online, ReWiND only requires a language description of the task, z_{new}. ReWiND rolls out $\pi(a \mid o_t, z_{\text{new}})$ and fine-tunes it on rewards coming from $R_\psi(o_{1:t}, z_{\text{new}})$. Like

prior work [10, 386], we assume access to a success signal during online RL. We use this signal to give r_{success} bonuses similar to in pre-training.[‡] Our online rewards $\hat{r}^{\text{on}}$ are:

$$\hat{r}^{\text{on}}_t = R_\psi(o_{1:t}, z) + r_{\text{success}} \cdot \mathbb{1}[\text{success at } t]. \tag{10.5}$$

See full implementation details in Appendix G.1.1 and pseudocode in Algorithm 9.

10.4 Experiments

Our experiments aim to study the efficacy of ReWiND as a reward learning pipeline, evaluate its ability to train robots to learn new tasks efficiently, and analyze its design choices and limitations. To this end, we organize our experiments to answer the following empirical questions, in order:

(**Q1**) **Rewards**: How well do ReWiND rewards correlate with task progress and success?

(**Q2**) **Policy Learning**: Can ReWiND quickly train policies for new tasks?

(**Q3**) **Ablations and Analysis**: Which ReWiND design decisions are most significant?

10.4.1 Q1: What Makes a Good Reward Function?

We repeat the desiderata from Section 10.3.1 that we set out to achieve with ReWiND: (1) generalization to new tasks, (2) rewards aligned with videos from *policy rollouts*, and (3) robustness to diverse inputs. We structure this section to demonstrate ReWiND's ability to satisfy these criteria.

We compare ReWiND-learned rewards against all relevant reward learning baselines from Section 10.2: **LIV** [221] is a robotics reward model pre-trained on EpicKitchens [73], we also fine-tune LIV on $\mathcal{D}_{\text{demos}}$ (**LIV-FT**); **RoboCLIP** [334] uses a pre-trained video language model, S3D [377] trained on HowTo100M [242],

[‡]Success bonuses can come from a human supervisor [214], learned function [106], or LLM [390]. Our experiments assume a human supervisor because manual resets are required regardless. While we could threshold $R_\psi(o_{1:t}, z)$ outputs to automatically determine success, unseen evaluation task reward ranges can vary, rendering this approach ineffective. Future work could integrate ReWiND with methods reducing human resets [387, 120] and automatic success detectors for truly autonomous RL.

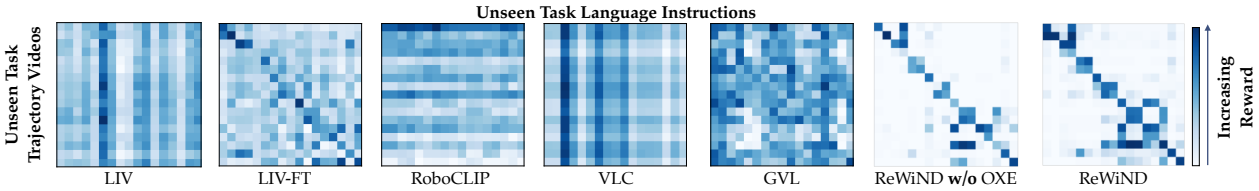


Figure 10.4: **Video-Language Reward Confusion Matrix.** For each unseen task, we compute rewards for all combinations of demonstration videos and language descriptions. ReWiND produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions. See Appendix G.4.1 for train task results.

to reward agents for language specified tasks; **Video-Language Critic (VLC)** [10] fine-tunes a VLM with a sequential ranking objective to encourage frames later in the video to have higher rewards. We train it on $\mathcal{D}_{\text{demos}}$; **Generative Value Learning (GVL)** [220] prompts a pre-trained Gemini LLM [345] with shuffled frames to predict per-frame progress.

We conduct our primary reward analysis using the simulated Meta-World benchmark [394] because it enables efficient collection of exemplar failed and partially successful rollout videos for analysis. Smaller-scale real-world reward experiments, **strongly aligned** with the results in simulation, are in Appendix G.4.3. $\mathcal{D}_{\text{demos}}$ here consists of 20 tasks with 5 expert demos each. For fair comparison, we include a variant of ReWiND trained without $\mathcal{D}_{\text{open-x}}$ (**ReWiND w/o OXE**). Results are evaluated on 17 unseen but related Meta-World tasks. We average metrics across 5 rollouts per task.

Generalization. We first evaluate how effectively each reward model distinguishes unseen tasks using confusion matrices of unseen task videos versus language instructions (Figure 10.4). Ideally, a clear blue diagonal indicates correct video-instruction pairs, with low (white) values elsewhere. ReWiND produces clearest disparity between the diagonal and off-diagonal elements, excelling even without OXE due to architectural choices aimed at generalization, i.e., first frame positional encodings and frozen pre-trained input embeddings.

Next, we evaluate how consistently rewards reflect progress over time in successful, unseen demonstrations. We report Pearson correlation (r) of each model's reward against time, and Spearman's rank

Table 10.1: **Combined Evaluation Metrics.** Comparison of reward models across three axes: (1) Demo Video Reward Alignment, (2) Policy Rollout Reward Ranking, and (3) Input Robustness.

Category	Metric	LIV	LIV-FT	RoboCLIP	VLC	GVL	ReWiND w/o OXE	ReWiND w/ OXE
(a) Demo Reward Alignment	$r \uparrow$	-0.03	0.55	0.01	0.64	0.52	0.67	**0.83**
	$\rho \uparrow$	-0.04	0.55	-0.01	0.62	0.57	0.64	**0.79**
(b) Policy Rollout Ranking	Rew. Order $\rho \uparrow$	-0.32	0.47	0.00	-0.18	0.32	0.76	**0.82**
	Rew. Diff. $\uparrow$	-0.16	0.26	0.06	-0.15	0.17	0.39	**0.41**
(c) Input Robustness	Avg. $\rho \uparrow$	0.03	0.27	0.00	0.60	0.58	0.55	**0.74**
	ρ Variance $\downarrow$	0.08	0.28	**0.00**	**0.00**	0.01	0.03	0.04

correlation (ρ), which, unlike r, captures monotonicity regardless of linearity. As shown in Table 10.1**(a)**, ReWiND again outperforms all baselines—achieving a **30%** relative improvement in r and **27%** in ρ over the best alternative (VLC).

Policy Rollout Reward Alignment. We also find that ReWiND can properly reward *failed* policy rollouts, which is important for rewarding RL policies on unseen tasks. For each task, we train an SAC [122] policy from scratch and use trajectories collected from various points of training to construct three evaluation video datasets: `failure`, `near-success`, and `success` containing failed trajectories, trajectories where the policy was close to the goal state but did not succeed, and successful trajectories, respectively. Each task has 2 trajectories of each type.

We evaluate each dataset's relative alignment *ranking* (measured by Spearman's ρ) with each reward model. For example, for a given task, if the average reward for a `failure` video is 0.1, a `near-success` video is 0.5, and `success` video is 0.9, then the rankings would be 1, 2, 3, respectively, where 3 corresponds to the best ranking. Thus, ρ over the rankings tells us how often the videos are correctly ranked. We report the ranking ρ in Table 10.1**(b)**. We also report the average *difference* between rewards for `success` with `near-success` and `near-success` with `failure` videos. Overall, likely due to *video rewinding*, ReWiND has a relative **74%** improvement in reward order and **58%** improvement in reward differences over the best baseline, LIV-FT. Additionally, we qualitatively demonstrate how these rankings translate into policy rollout rewards in Appendix Figure G.1 by plotting per-frame reward curve predictions of ReWiND against reward baselines for an unsuccessful policy rollout.

Robustness to Varied Inputs. Finally, we demonstrate ReWiND's robustness to diverse instructions. For each evaluation task, we manually create three additional language instructions (without prior knowledge of ReWiND's performance), resulting in four total instructions per task. For example, "close the door" is an original instruction, and we add "shut the door." Each set of instructions is paired with a single demonstration video, and we compare the reward models by measuring their average Spearman's rank correlation (ρ) and output variance across these instructions in Table 10.1**(c)**. Higher variance indicates lower robustnes. Again, ReWiND outperforms baselines, achieving the highest average correlation (0.74), **23%** better than VLC, and near-zero variance, even without OXE training—likely aided by our instruction augmentation approach (Section 10.3.1.2). RoboCLIP and VLC show near-zero variance but achieve significantly lower correlation scores.

So far, our results demonstrate that **ReWiND significantly outperforms all image-language-conditioned reward baselines** in terms of **generalization**, rewarding **policy rollouts**, and input **robustness**. We next demonstrate how these results translate into sample-efficient policy learning.

10.4.2 Q2: Learning New Tasks with RL

Simulation. We use the Meta-World simulation benchmark [394], where we pre-train reward models and policies on 20 tasks, each with 5 per-task demos collected from a scripted policy. We evaluate on 8 unseen tasks in Meta-World, chosen for reasonable initial policy rollout behaviors, across 3 seeds each. We compare ReWiND against the 2 language-conditioned reward model baselines that performed best in reward alignment (**VLC**) and policy rollout rankings (**LIV-FT**) from the reward analysis in Section 10.4.1. We also compare against **Sparse**, which pre-trains and

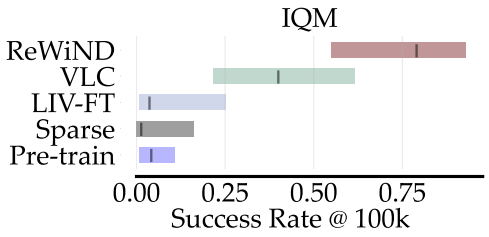


Figure 10.5: **Meta-World final performance.** We plot inter-quartile means (IQMs) of success rates after 100k environment steps on 8 unseen tasks in Meta-World. ReWiND achieves 79%.

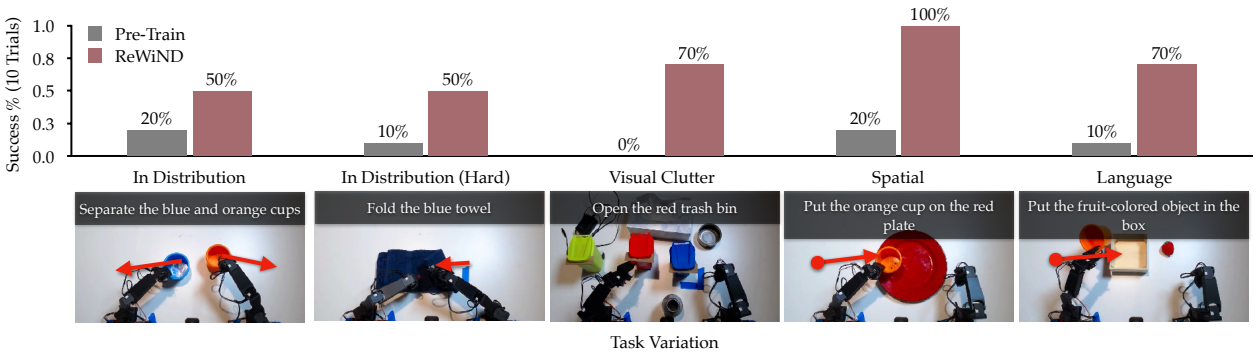


Figure 10.6: **Real-robot RL.** We present results on the Koch bimanual arms across in-distribution tasks and visual, spatial, and linguistic generalization tasks. Online RL with ReWiND improves a pre-trained policy by an absolute 56% across all five tasks.

fine-tunes on only the sparse success reward bonus, and **Pre-train**, which pre-trains on sparse reward and is evaluated zero-shot on new tasks. All baselines are image, proprioception (x, y, z, gripper), and language conditioned. Each method uses the same policy pre-training and RL procedure as ReWiND as outlined in Section 10.3.2, and is trained online for 100k timesteps. See Appendix G.2 for environment and policy training details.

As recommended by Agarwal et al. [4], we report the interquartile mean (IQM) and 95% confidence intervals computed over all task success rates at 100k environment steps in Figure 10.5. Sparse reward fine-tuning and Pre-train (no fine-tuning) result in near-zero success rates, highlighting the difficulty of image-based new task learning under limited data. In fact, Sparse reward fine-tuning, which relies purely on a sparse success bonus, performs worse than Pre-train after fine-tuning. Meanwhile, ReWiND achieves an IQM success rate of **79%**, a **97.5%** improvement over the best baseline, VLC, demonstrating that ReWiND effectively enables the policy to learn new tasks in Meta-World. These results are well-aligned with our reward analysis in Section 10.4.1, demonstrating how they correlate with policy learning performance. ReWiND is also more sample-efficient at timesteps less than 100k; see extended discussion in Appendix G.4.2 and sample efficiency curves in Figure G.7i.

Real-World Robot Learning. We conduct real-world tabletop manipulation experiments with a bimanual Koch v1.1 robot arm setup [47]. We use 5 demos to train the reward function, but 10 for the policy,

as we found policy learning to be a bottleneck on this difficult robot embodiment. Across five tasks, we demonstrate in Figure 10.6 that an hour of real-world reinforcement learning with ReWiND improves the success rate over the base pre-trained policy from an average 12% success rate to 68%, a **5X** improvement. RL for an hour of real-world experiment time corresponds to 50k environment steps with our parallelized codebase that trains the policy while an older checkpoint gathers data in the environment to avoid any training wait time. We select diverse tasks that demonstrate real-world improvement based on generalization metrics defined in prior work [17, 108] on: an in-distribution task, `separate the blue and orange cups`; an in-distribution *difficult* task, `fold the blue towel`; an unseen task in terms of large amounts of *visual* clutter, `open the red trash bin`; an unseen task in terms of spatial relationships between objects requiring *new action sequences*, `put the orange cup on the red plate`; and an unseen task in terms of *language* input, `put the fruit-colored object in the box`. Overall, ReWiND enables real-world reinforcement learning on unseen tasks without requiring new demonstrations, improving over the pre-trained pre-trained policy, and outperforms the best baseline from simulation, VLC. See Appendix G.3.1 for real-world experiment details and Figure G.5 for policy rollout examples.

Q3: Ablations and Analysis. ReWiND's ability to teach policies unseen tasks comes from achieving the three desiderata listed earlier, namely generalization, providing accurate rewards for policy rollout failures, and input robustness. We demonstrate that each component of ReWiND contributes to at least one of each desiderata in Appendix G.5 where we ablate instruction augmentation, video rewinding, the use of $\mathcal{D}_{\text{demos}}$, and first-frame positional embeddings.

Concluding Statement. In conclusion, our experiments demonstrated ReWiND's effectiveness as a reward function for policy learning through detailed reward analyses and its effectiveness as a framework for sample-efficient robot learning of unseen tasks, both in simulation and on a real bimanual robot. Finally, we thoroughly discuss limitations and failure cases of ReWiND in Section 10.5.

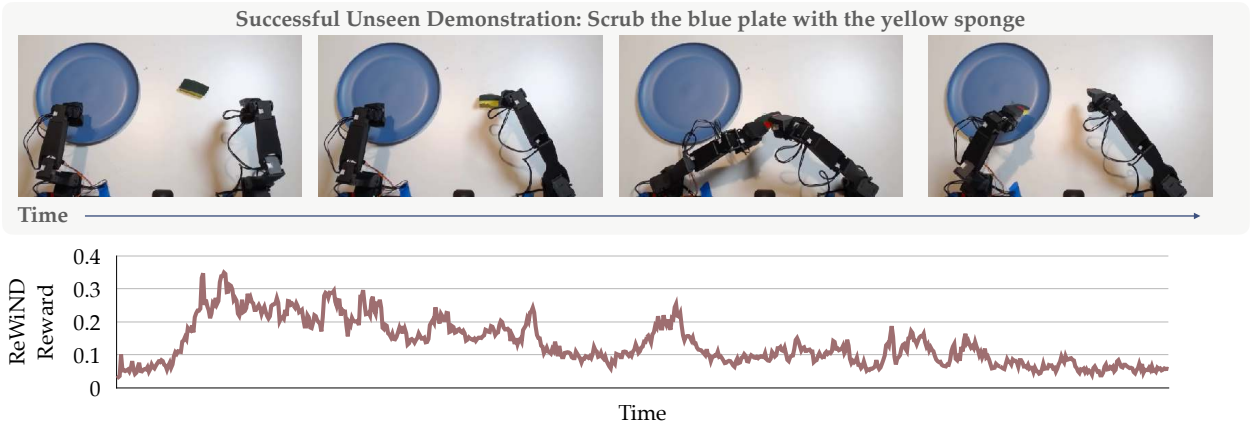


Figure 10.7: **ReWiND Failure Example.** We collect a demonstration of the Koch arms picking up a sponge, handing it over, and scrubbing a plate. We find that there is poor reward alignment to this successful demonstration, likely due to the lack of bimanual data in Open-X, occlusion of the sponge, and poor camera viewpoints.

10.5 Limitations

Reward Analysis. One of the limitations of ReWiND lies in its inherent tradeoff using pre-trained vision and language embeddings. We do not fine-tune these embeddings because our assumed demonstration dataset $\mathcal{D}_{\text{demos}}$ is very small, and in early experiments, we found that fine-tuning sometimes hurts generalization performance. Not fine-tuning may result in underfitting certain tasks, particular to robotics, on which the pre-trained vision and language models were not trained. In Figure 10.7, we visualize an example of dish scrubbing which ReWiND does not perform well on even though similar linguistic tasks exist in the Open-X dataset. This poor result is likely due to Open-X not containing any bimanual data or partial occlusion due to the camera viewpoint. Future work that pre-trains with even more robotics data or incorporates intermediate representations or objectives with large-scale pre-training on internet data (e.g., Li et al. [192] and Team et al. [346]) could allow fine-tuning the input embeddings to ensure they can better fit the $\mathcal{D}_{\text{demos}}$.

Initial Policy Performance. Finally, we use a relatively simple policy architecture that is trained from scratch for each of our domains. We expect better performance by combining ReWiND with stronger

policy architectures capable of ingesting more data (e.g., pre-trained vision-language-action models) that have better *zero-shot performance* on new tasks to enable even more sample-efficient learning irrespective of reward function.

In fact, we confirmed experimentally that initial zero-shot performance was strongly indicative of how well the policy will learn a new task in our real-world experiments. For example, ReWiND does not help a policy that confidently performs the wrong task. If the ReWiND reward function could be combined with stronger policies that are easy to learn online in the loop, we hope it will enable learning of many more difficult new tasks.

However, the best way to fine-tune these models with online rewards remains an open challenge [254, 116]. One bottleneck is simply that, even with low-rank adaptation techniques that prior work found to help train large policy architectures more efficiently [210, 116, 165], fine-tuning these models takes a lot of compute and real-world time that makes real-world online learning with reward difficult. We plan to investigate blending ReWiND approaches with such policy architectures in the future.

Resets. ReWiND, in its current form, requires a human operator to perform resets of the environment. This assumption prevents ReWiND from being fully autonomous. However, recent reset-free RL works [387, 120, 246, 390] demonstrate promising solutions to address the need for humans to supervise learning. Regardless, human resets remain a roadblock to autonomous learning that is difficult to address in the real world [245].

Success Detection. Another limitation comes from requiring success detection for the reward bonus and terminating policy rollouts upon success. We add a success bonus (detailed in Section 10.3.2) to account for potential noisy rewards and imperfect success detection by the reward model, given that a human is already monitoring to reset the environment, and terminate the rollout upon success. We visualize examples in Figure 10.7 of imperfect reward predictions in unseen tasks with lower than expected final rewards given

to the last successful observation. These examples demonstrate the need for a success bonus, and we saw similar or worse examples across all reward learning approaches evaluated in Section 10.4.1. Methods such as those introduced by Ye et al. [390], Zhou et al. [415], and Yang et al. [387], which utilize VLMs as success detectors, can remove the need for human supervision during the online phase of ReWiND when combined with reset-free RL. In future work, we plan to investigate the combination of ReWiND with reset-free approaches and automatic success detection for truly autonomous learning.

Chapter 11

Conclusions

11.1 Further advancing real-world robot learning

My work with real-world robots [407, 404, 330, 402, 192, 385] and focus on adapting robots with RL in the real world have highlighted two major remaining challenges: (1) achieving strong pre-trained policy generalization and (2) reducing the need for human-driven environment resets. These open problems represent significant hurdles to overcome in future research.

Policy Generalization. For autonomous learning, policies must perform well from the start to avoid unsafe or erratic behavior that necessitates human monitoring while learning [401]. Advances in LPTMs and large-scale robotics datasets [67] make LPTM-based policies promising for generalizing well to new tasks. I propose taking advantage of *hierarchy* to train data-efficient VLAs where a higher-level VLM can be trained on various sources of cross-domain data for better generalization than standard VLAs, while low-level specialist policies can execute precise actions with high-frequency execution. In work accepted to ICLR 2025 [192] that I talked about in Chapter 5, this architecture outperformed monolithic VLAs like OpenVLA [165]. Building on this intuition, I plan to develop a hierarchical VLA where the high-level VLM *reduces input complexity* for a low-level policy by masking irrelevant objects and predicting a high-level path for the robot arm. This VLA will be trained via automatic data labeling using powerful vision LPTMs to label *arbitrary* visual robotics datasets. Reduced input complexity will allow the low-level policy to

generalize much better to varied image inputs, allowing training even on simulation data. Further in the future, I plan on extending my continual learning [210] and offline RL techniques [404, 407], both of which were applied to large transformer policies, to fine-tune these large VLAs without overfitting on incoming online interaction data to ensure stable, sample-efficient autonomous learning.

Environment Resets. Finally, real-world autonomous learning is also hampered by the need for humans to reset the environment after each episode of data collection [245]. Prior work has attempted to reduce the need for resets [92, 246, 120] and has made great advances towards reset-free learning, but they still can require many manual human resets while learning or human-designed curricula for each environment which is not scalable. To drastically reduce human reset frequency, I propose an alternative *evaluation*-based approach where the robot predicts its performance and automatically designs a task curriculum that includes built-in reset behaviors. For example, if the robot can reliably solve the task "stand up the cereal box," it can confidently attempt "put the cereal box in the cupboard," knowing it can reset by standing the box up if dropped. This procedure can be combined with LPTMs to automatically ensure that the proposed tasks and associated reset tasks are sensible. This approach should vastly reduce the required human supervision needed for autonomous learning. I plan on working on this problem in the future.

11.2 Expanding to other robotics domains

The methods proposed in this thesis were primarily evaluated on tabletop manipulation tasks, both in simulation and the real world. Therefore, one interesting avenue of future research is to expand these approaches to other domains, such as robotic locomotion. In theory, given the same dataset assumptions, the algorithms in all chapters except Chapter 7, which was designed for interpreting human hand demonstrations for robot manipulation, can be applied to other robotics tasks. However, these dataset assumptions may not directly translate.

For example, Chapter 3 assumes language labels for sequences of low-level actions in the dataset, but this may not be a practical assumption when training robot locomotion policies as it may be hard to label minute differences in various locomotion gaits and styles. However, many of the takeaways and techniques of each part are still applicable:

- **Pre-Training Part I**: LPTMs can help us label high-level behaviors, i.e., skills, even on unlabeled data (Chapter 4). These skills can also be chained together with offline RL (Chapter 3) so that robots can more easily learn new long-horizon tasks. Meanwhile, we can use high-level guidance from LPTMs fine-tuned on intermediate representations, such as paths, on web-scale robotics data (Chapter 5) to achieve superior visual and semantic generalization in our pre-trained policies.

- **Adapting with Human Supervision Part II**: We can use LPTMs to help pre-train on lots of available robotics data and fine-tune with low-rank adapters to new tasks given some human guidance in the form of human demonstrations (Chapter 6). We can also use LPTMs to interpret human guidance to help find relevant robotics data from offline data to train our robot for a specific task we want it to adapt to (Chapter 7).

- **Adapting with Minimal Supervision Part III**: LPTMs can help robots figure out *which tasks are important* and practice them after being deployed to new environments. For example, after deploying a locomotion robot to a new grassy environment where it's unable to run quickly, an LLM can help the robot practice running fast and jumping over obstacles (Chapter 8). Then, we can use LPTMs to help provide dense rewards for semantically meaningful skills that are difficult to write reward functions for (Chapter 9 and Chapter 10).

11.3 Concluding Statement.

Throughout my thesis, I proposed algorithms that enable scalable policy adaptation via LPTM guidance for (1) pre-training, (2) online learning with human guidance, and (3) autonomous learning. By combining these approaches with future work addressing policy generalization and environment resets, I plan to make further progress toward enabling truly autonomous robots that require minimal human supervision as they adapt to new tasks and settings. I hope you have enjoyed reading this thesis as much as I did working on the projects in it. Despite all of the trials and tribulations we all face in our PhDs, I felt this was a very rewarding and humbling journey that I am glad to have experienced.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2004.

[2] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. "Variational Option Discovery Algorithms". In: *arXiv* (2018).

[3] OpenAI Josh Achiam et al. "GPT-4 Technical Report". In: *arxiv preprint*. 2023. URL: https://arxiv.org/pdf/2303.08774.

[4] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. "Deep reinforcement learning at the edge of the statistical precipice". In: *NeurIPS*. 2021.

[5] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. "Reincarnating reinforcement learning: Reusing prior computation to accelerate progress". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 28955–28971.

[6] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. "Intrinsic dimensionality explains the effectiveness of language model fine-tuning". In: *arXiv preprint arXiv:2012.13255* (2020).

[7] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. "Do As I Can and Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691*. 2022.

[8] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. "{OPAL}: Offline Primitive Discovery for Accelerating Offline Reinforcement Learning". In: *International Conference on Learning Representations*. 2021.

[9] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. "OPAL: Offline Primitive Discovery for Accelerating Offline Reinforcement Learning". In: *arXiv preprint arXiv:2010.13611* (2020).

[10] Minttu Alakuijala, Reginald McLean, Isaac Woungang, Nariman Farsad, Samuel Kaski, Pekka Marttinen, and Kai Yuan. “Video-Language Critic: Transferable Reward Functions for Language-Conditioned Robotics”. In: *Transactions on Machine Learning Research (TMLR)*. 2025.

[11] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. “Concrete problems in AI safety”. In: *arXiv preprint arXiv:1606.06565* (2016).

[12] John R. Anderson. “Acquisition of cognitive skill.” In: *Psychological Review* 89 (1982), pp. 369–406.

[13] Jacob Andreas, Dan Klein, and Sergey Levine. “Learning with latent language”. In: *NAACL*. 2017.

[14] Jacob Andreas, Dan Klein, and Sergey Levine. “Modular multitask reinforcement learning with policy sketches”. In: *ICML*. 2017.

[15] Jacob Andreas, Dan Klein, and Sergey Levine. “Modular multitask reinforcement learning with policy sketches”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 166–175.

[16] Xavier Anguera, Simon Bozonnet, Nicholas Evans, Corinne Fredouille, Gerald Friedland, and Oriol Vinyals. “Speaker diarization: A review of recent research”. In: *IEEE Transactions on audio, speech, and language processing* 20.2 (2012), pp. 356–370.

[17] Abrar Anwar, Rohan Gupta, and Jesse Thomason. “Contrast Sets for Evaluating Language-Guided Robot Policies”. In: *Conference on Robot Learning (CoRL)*. 2024.

[18] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[19] Pierre-Luc Bacon, Jean Harb, and Doina Precup. “The Option-Critic Architecture.” In: *AAAI*. 2017.

[20] Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. “Affordances from Human Videos as a Versatile Representation for Robotics”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023.

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

[22] Andrea Bajcsy, Dylan P Losey, Marcia K O’Malley, and Anca D Dragan. “Learning from physical human corrections, one feature at a time”. In: *International Conference on Human-Robot Interaction (HRI)*. 2018.

[23] Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. *Efficient Online Reinforcement Learning with Offline Data*. 2023. arXiv: 2302.02948 [cs.LG].

[24] Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. “Efficient Online Reinforcement Learning with Offline Data”. In: *International Conference on Machine Learning (ICML)*. 2023.

[25] GEORGE A. BEKEY. “On autonomous robots”. In: *The Knowledge Engineering Review* 13.2 (1998), pp. 143–146. DOI: 10.1017/S0269888998002033.

[26] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. “HYDRA: Hybrid Robot Actions for Imitation Learning”. In: *Conference on Robot Learning (CoRL)*. 2023.

[27] Homanga Bharadhwaj, Roozbeh Mottaghi, Abhinav Gupta, and Shubham Tulsiani. “Track2Act: Predicting Point Tracks from Internet Videos enables Diverse Zero-shot Robot Manipulation”. In: *arXiv preprint arXiv:2405.01527* (2024).

[28] Christopher M Bishop. “Mixture density networks”. In: (1994).

[29] Erdem Biyik, Nicolas Huynh, Mykel J. Kochenderfer, and Dorsa Sadigh. “Active Preference-Based Gaussian Process Regression for Reward Learning”. In: *Robotics: Science and Systems (RSS)*. 2020.

[30] Erdem Biyik, Malayandi Palan, Nicholas C. Landolfi, Dylan P. Losey, and Dorsa Sadigh. “Asking Easy Questions: A User-Friendly Approach to Active Reward Learning”. In: *Proceedings of the 3rd Conference on Robot Learning (CoRL)*. 2019.

[31] Erdem Bıyık, Dylan P Losey, Malayandi Palan, Nicholas C Landolfi, Gleb Shevchuk, and Dorsa Sadigh. “Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences”. In: *The International Journal of Robotics Research* 41.1 (2022), pp. 45–67.

[32] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. “pi_0: A Vision-Language-Action Flow Model for General Robot Control”. In: *arXiv preprint arXiv:2410.24164* (2024).

[33] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. “RoboCat: A Self-Improving Foundation Agent for Robotic Manipulation”. In: *arXiv preprint arXiv:2306.11706* (2023).

[34] Stevo Bozinovski and Ante Fulgosi. “The influence of pattern similarity and transfer learning upon training of a base perceptron b2”. In: *Proceedings of Symposium Informatica*. Vol. 3. 1976, pp. 121–126.

[35] Satchuthananthavale RK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. “Reinforcement learning for mapping instructions to actions”. In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 2009, pp. 82–90.

[36] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. “RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control”. In: *arXiv preprint arXiv:2307.15818*. 2023.

[37] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *Conference on Robot Learning (CoRL)*. 2023.

[38] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. "RT-1: Robotics Transformer for Real-World Control at Scale". In: *arXiv preprint arXiv:2212.06817*. 2022.

[39] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. "RT-1: Robotics Transformer for Real-World Control at Scale". In: *Robotics: Science and Systems (RSS)*. 2023.

[40] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *Conference on robot learning*. PMLR. 2023, pp. 287–318.

[41] Daniel S. Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. "Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations". In: *International Conference on Machine Learning (ICML)*. 2019.

[42] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[43] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. 2020. DOI: 10.48550/ARXIV.2005.14165.

[44] Minwoo Byeon, Beomhee Park, Haecheon Kim, Sungjun Lee, Woonhyuk Baek, and Saehoon Kim. *COYO-700M: Image-Text Pair Dataset*. https://github.com/kakaobrain/coyo-dataset. 2022.

[45] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. "Scaling data-driven robotics with reward sketching and batch reinforcement learning". In: *Robotics: Science and Systems (RSS)*. 2020.

[46] Massimo Caccia, Jonas Mueller, Taesup Kim, Laurent Charlin, and Rasool Fakoor. "Task-Agnostic Continual Reinforcement Learning: Gaining Insights and Overcoming Challenges". In: *Conference on Lifelong Learning Agents*. 2023.

[47] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, and Thomas Wolf. *LeRobot: State-of-the-art Machine Learning for Real-World Robotics in Pytorch*. https://github.com/huggingface/lerobot. 2024.

[48] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. "On tiny episodic memories in continual learning". In: *arXiv preprint arXiv:1902.10486* (2019).

[49] Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jacob Varley, Alex Irpan, Benjamin Eysenbach, Ryan C Julian, Chelsea Finn, and Sergey Levine. "Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 1518–1528. URL: https://proceedings.mlr.press/v139/chebotar21a.html.

[50] Annie S Chen, Suraj Nair, and Chelsea Finn. "Learning Generalizable Robotic Reward Functions from" In-The-Wild" Human Videos". In: *RSS* (2021). arXiv: 2103.16817 [cs.RO].

[51] Annie S. Chen, Suraj Nair, and Chelsea Finn. "Learning Generalizable Robotic Reward Functions from "In-The-Wild" Human Videos". In: *Robotics: Science and Systems (RSS)*. 2021.

[52] Baiming Chen, Zuxin Liu, Jiacheng Zhu, Mengdi Xu, Wenhao Ding, Liang Li, and Ding Zhao. "Context-aware safe reinforcement learning for non-stationary environments". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 10689–10695.

[53] Lawrence Yunliang Chen, Simeon Adebola, and Ken Goldberg. *Berkeley UR5 Demonstration Dataset*. https://sites.google.com/view/berkeley-ur5/home. 2023.

[54] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. "Decision transformer: Reinforcement learning via sequence modeling". In: *Advances in neural information processing systems* 34 (2021), pp. 15084–15097.

[55] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. "Adaptformer: Adapting vision transformers for scalable visual recognition". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 16664–16678.

[56] Zoey Chen, Sho Kiami, Abhishek Gupta, and Vikash Kumar. "GenAug: Retargeting behaviors to unseen situations via Generative Augmentation". In: *RSS*. 2023.

[57] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*. Ed. by Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu. 2023. DOI: 10.15607/RSS.2023.XIX.026.

[58] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. *PaLM: Scaling Language Modeling with Pathways*. 2022. DOI: 10.48550/ARXIV.2204.02311.

[59] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. *Deep reinforcement learning from human preferences*. 2023. arXiv: 1706.03741 [stat.ML].

[60] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep reinforcement learning from human preferences". In: *NeurIPS*. 2017.

[61] Petros Christodoulou. *Soft Actor-Critic for Discrete Action Settings*. 2019. arXiv: 1910.07207 [cs.LG].

[62] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. "Scaling Instruction-Finetuned Language Models". In: (2022). arXiv: 2210.11416 [cs.LG].

[63] Geoffrey Cideron, Mathieu Seurin, Florian Strub, and Olivier Pietquin. "Higher: Improving instruction following with hindsight generation for experience replay". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2020, pp. 225–232.

[64] Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, F Peter Dominey, and Pierre-Yves Oudeyer. "Language as a Cognitive Tool to Imagine Goals in Curiosity Driven Exploration". In: *NeurIPS 2020* (2020).

[65] Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Dominey, and Pierre-Yves Oudeyer. "Language as a cognitive tool to imagine goals in curiosity driven exploration". In: *Advances in Neural Information Processing Systems* (2020).

[66] Cédric Colas, Laetitia Teodorescu, Pierre-Yves Oudeyer, Xingdi Yuan, and Marc-Alexandre Côté. "Augmenting Autotelic Agents with Large Language Models". In: *Conference on Lifelong Learning Agents*. 2023.

[67] Open X-Embodiment Collaboration et al. "Open X-Embodiment: Robotic Learning Datasets and RT-X Models". In: *International Conference on Robotics and Automation (ICRA)*. 2024.

[68] Erwin Coumans and Yunfei Bai. *Pybullet, a python module for physics simulation for games, robotics and machine learning*. 2016.

[69] Yuchen Cui and Scott Niekum. "Active reward learning from critiques". In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[70] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. "Can Foundation Models Perform Zero-Shot Task Specification For Robot Manipulation?" In: *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*. Ed. by Roya Firoozi, Negar Mehr, Esen Yel, Rika Antonova, Jeannette Bohg, Mac Schwager, and Mykel Kochenderfer. Vol. 168. Proceedings of Machine Learning Research. PMLR, June 2022, pp. 893–905. URL: https://proceedings.mlr.press/v168/cui22a.html.

[71] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. "Can foundation models perform zero-shot task specification for robot manipulation?" In: *Learning for Dynamics and Control Conference (L4DC)*. 2022.

[72] Murtaza Dalal, Deepak Pathak, and Ruslan Salakhutdinov. "Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives". In: *NeurIPS*. 2021.

[73] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Jian Ma, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. "Rescaling Egocentric Vision: Collection, Pipeline and Challenges for EPIC-KITCHENS-100". In: *International Journal of Computer Vision (IJCV)* 130 (2022), pp. 33–55.

[74] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. "RoboNet: Large-scale multi-robot learning". In: *CoRL* (2019).

[75] Shivin Dass, Karl Pertsch, Hejia Zhang, Youngwoon Lee, Joseph J. Lim, and Stefanos Nikolaidis. *PATO: Policy Assisted TeleOperation for Scalable Robot Data Collection*. 2023. arXiv: 2212.04708 [cs.RO].

[76] Shivin Dass, Jullian Yapeter, Jesse Zhang, Jiahui Zhang, Karl Pertsch, Stefanos Nikolaidis, and Joseph J. Lim. *CLVR Jaco Play Dataset*. Version 1.0.0. 2023. URL: https://github.com/clvrai/clvr_jaco_play_dataset.

[77] Matt Deitke et al. "Molmo and PixMo: Open Weights and Open Data for State-of-the-Art Multimodal Models". In: *arXiv preprint arXiv:2409.17146* (2024).

[78] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[79] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale". In: *arXiv preprint arXiv:2208.07339* (2022).

[80] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805.

[81] Thomas G Dietterich, Lorien Pratt, and Sebastian Thrun. "Special issue on inductive transfer". In: *Machine Learning* 28.1 (1997).

[82] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. "Tapir: Tracking any point with per-frame initialization and temporal refinement". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 10061–10072.

[83] David H Douglas and Thomas K Peucker. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature". In: *Cartographica* 10.2 (1973), pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727. eprint: https://doi.org/10.3138/FM57-6770-U75U-7727.

[84] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, et al. "PaLM-E: An Embodied Multimodal Language Model". In: *arXiv preprint arXiv:2303.03378*. 2023.

[85] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. "PaLM-E: An Embodied Multimodal Language Model". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 8469–8488.

[86] Maximilian Du, Suraj Nair, Dorsa Sadigh, and Chelsea Finn. "Behavior Retrieval: Few-Shot Imitation Learning by Querying Unlabeled Datasets". In: *Robotics: Science and Systems*. 2023.

[87] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. *Guiding Pretraining in Reinforcement Learning with Large Language Models*. 2023. DOI: 10.48550/ARXIV.2302.06692.

[88] Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. "Manipulate-Anything: Automating Real-World Robots using Vision-Language Models". In: *arXiv preprint arXiv:2406.18915* (2024).

[89] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. "RL ^{2}: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[90] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. *Bridge Data: Boosting Generalization of Robotic Skills with Cross-Domain Datasets*. 2021. arXiv: 2109.13396 [cs.RO].

[91] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. "Bridge data: Boosting generalization of robotic skills with cross-domain datasets". In: *RSS*. 2022.

[92] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. "Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=S1vuO-bCW.

[93] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. "Diversity is All You Need: Learning Skills without a Reward Function". In: *ICLR*. 2019.

[94] Benjamin Eysenbach, Tianjun Zhang, Ruslan Salakhutdinov, and Sergey Levine. *Contrastive Learning as Goal-Conditioned Reinforcement Learning*. 2022. DOI: 10.48550/ARXIV.2206.07568.

[95] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. "Meta-Q-Learning". In: *International Conference on Learning Representations*. 2020.

[96] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. "Meta-Q-Learning". In: *International Conference on Learning Representations*. 2020.

[97] Rasool Fakoor, Jonas W Mueller, Kavosh Asadi, Pratik Chaudhari, and Alexander J Smola. "Continuous doubly constrained batch reinforcement learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 11260–11273.

[98] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. "MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge". In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022.

[99] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *ICML* (2017).

[100] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *International Conference on Machine Learning (ICML)*. 2016.

[101] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. "Motion Policy Networks". In: *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*. Ed. by Karen Liu, Dana Kulic, and Jeffrey Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2022, pp. 967–977. URL: https://proceedings.mlr.press/v205/fishman23a.html.

[102] P.M. Fitts and M.I. Posner. *Human Performance*. Basic concepts in psychology series. Brooks/Cole Publishing Company, 1967. ISBN: 9780134452470.

[103] Haotian Fu, Shangqun Yu, Michael Littman, and George Konidaris. "Model-based Lifelong Reinforcement Learning with Bayesian Exploration". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 32369–32382.

[104] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. "D4rl: Datasets for deep data-driven reinforcement learning". In: *arXiv preprint arXiv:2004.07219* (2020). arXiv: 2004.07219 [cs.LG].

[105] Justin Fu, Katie Luo, and Sergey Levine. "Learning robust rewards with adversarial inverse reinforcement learning". In: *International Conference on Learning Representations (ICLR)*. 2018.

[106] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. "Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition". In: *NeurIPS*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. 2018.

[107] Scott Fujimoto, David Meger, and Doina Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *International Conference on Machine Learning*. 2019, pp. 2052–2062.

[108] Jensen Gao, Suneel Belkhale, Sudeep Dasari, Ashwin Balakrishna, Dhruv Shah, and Dorsa Sadigh. "A Taxonomy for Evaluating Generalist Robot Policies". In: *arXiv preprint arXiv:2503.01238* (2025).

[109] Yuying Ge, Annabella Macaluso, Li Erran Li, Ping Luo, and Xiaolong Wang. *Policy Adaptation from Foundation Model Feedback*. 2023. arXiv: 2212.07398 [cs.LG].

[110] Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. "RVT2: Learning Precise Manipulation from Few Demonstrations". In: *RSS* (2024).

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

[112] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, Dhruv Batra, Vincent Cartillier, Sean Crane, Tien Do, Morrie Doulaty, Akshay Erapalli, Christoph Feichtenhofer, Adriano Fragomeni, Qichen Fu, Abrham Gebreselasie, Cristina Gonzalez, James Hillis, Xuhua Huang, Yifei Huang, Wenqi Jia, Weslie Khoo, Jachym Kolar, Satwik Kottur, Anurag Kumar, Federico Landini, Chao Li, Yanghao Li, Zhenqiang Li, Karttikeya Mangalam, Raghava Modhugu, Jonathan Munro, Tullie Murrell, Takumi Nishiyasu, Will Price, Paola Ruiz Puentes, Merey Ramazanova, Leda Sari, Kiran Somasundaram, Audrey Southerland, Yusuke Sugano, Ruijie Tao, Minh Vo, Yuchen Wang, Xindi Wu, Takuma Yagi, Ziwei Zhao, Yunyi Zhu, Pablo Arbelaez, David Crandall, Dima Damen, Giovanni Maria Farinella, Christian Fuegen, Bernard Ghanem, Vamsi Krishna Ithapu, C. V. Jawahar, Hanbyul Joo, Kris Kitani, Haizhou Li, Richard Newcombe, Aude Oliva, Hyun Soo Park, James M. Rehg, Yoichi Sato, Jianbo Shi, Mike Zheng Shou, Antonio Torralba, Lorenzo Torresani, Mingfei Yan, and Jitendra Malik. *Ego4D: Around the World in 3,000 Hours of Egocentric Video*. 2022. arXiv: 2110.07058 [cs.CV].

[113] Karol Gregor, George Papamakarios, Frederic Besse, Lars Buesing, and Theophane Weber. "Temporal Difference Variational Auto-Encoder". In: (2019).

[114] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. "Variational Intrinsic Control". In: *arXiv* abs/1611.07507 (2016).

[115] Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. *RT-Trajectory: Robotic Task Generalization via Hindsight Trajectory Sketches*. 2023. arXiv: 2311.01977 [cs.RO].

[116] Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. "Improving Vision-Language-Action Model with Online Reinforcement Learning". In: *arXiv preprint arXiv:2501.16664* (2025). arXiv: 2501.16664 [cs.RO].

[117] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. "Relay Policy Learning: Solving Long Horizon Tasks via Imitation and Reinforcement Learning". In: *Conference on Robot Learning (CoRL)* (2019).

[118] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning". In: *CoRL* (2019). arXiv: 1910.11956 [cs.LG].

[119] Abhishek Gupta, Corey Lynch, Brandon Kinman, Garrett Peake, Sergey Levine, and Karol Hausman. "Demonstration-Bootstrapped Autonomous Practicing via Multi-Task Reinforcement Learning". In: *arXiv* (2022).

[120] Abhishek Gupta, Justin Yu, Tony Z. Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. "Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention". In: *International Conference on Robotics and Automation (ICRA)*. 2021.

[121] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Markus Wulfmeier, Jan Humplik, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, Michael Bloesch, Kristian Hartikainen, Arunkumar Byravan, Leonard Hasenclever, Yuval Tassa, Fereshteh Sadeghi, Nathan Batchelor, Federico Casarini, Stefano Saliceti, Charles Game, Neil Sreendra, Kushal Patel, Marlon Gwira, Andrea Huber, Nicole Hurley, Francesco Nori, Raia Hadsell, and Nicolas Heess. *Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning*. 2023. arXiv: 2304.13653.

[122] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)*. 2018.

[123] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *ICML* (2018).

[124] Douglas Hackett, James Pippine, Adam Watson, Charles Sullivan, and Gill Pratt. "An Overview of the DARPA Autonomous Robotic Manipulation (ARM) Program". In: *Journal of the Robotics Society of Japan* 31 (June 2013), pp. 326–329. DOI: 10.7210/jrsj.31.326.

[125] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. "Inverse reward design". In: *Advances in neural information processing systems* 30 (2017).

[126] Siddhant Haldar, Jyothish Pari, Anant Rai, and Lerrel Pinto. "Teach a Robot to FISH: Versatile Imitation from One Minute of Demonstrations". In: *arXiv preprint arXiv:2303.01497* (2023).

[127] Siddhant Haldar, Zhuoran Peng, and Lerrel Pinto. "Baku: An efficient transformer for multi-task policy learning". In: *Neural Information Processing Systems* (2024).

[128] Siddhant Haldar and Lerrel Pinto. "Point Policy: Unifying Observations and Actions with Key Points for Robot Manipulation". In: *arXiv preprint arXiv:2502.20391* (2025).

[129] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. "Dextreme: Transfer of agile in-hand manipulation from simulation to reality". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 5977–5984.

[130] Nicklas Hansen, Zhecheng Yuan, Yanjie Ze, Tongzhou Mu, Aravind Rajeswaran, Hao Su, Huazhe Xu, and Xiaolong Wang. "On pre-training for visuo-motor control: Revisiting a learning-from-scratch baseline". In: *arXiv preprint arXiv:2212.05749* (2022).

[131] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. "Learning an embedding space for transferable robot skills". In: *ICLR*. 2018.

[132] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. “Towards a Unified View of Parameter-Efficient Transfer Learning”. In: *International Conference on Learning Representations*. 2022.

[133] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *CVPR*. 2016, pp. 770–778.

[134] Joey Hejna and Dorsa Sadigh. “Few-Shot Preference Learning for Human-in-the-Loop RL”. In: *Conference on Robot Learning (CoRL)*. 2022.

[135] Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J. Lim. “FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation”. In: *Robotics: Science and Systems*. 2023.

[136] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. “Deep q-learning from demonstrations”. In: *AAAI*. 2018.

[137] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. “beta-vae: Learning basic visual concepts with a constrained variational framework”. In: *ICLR*. 2016.

[138] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *NeurIPS*. 2016.

[139] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. *Training Compute-Optimal Large Language Models*. 2022. DOI: 10.48550/ARXIV.2203.15556.

[140] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. “Parameter-efficient transfer learning for NLP”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2790–2799.

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

[142] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. “Lora: Low-rank adaptation of large language models.” In: *International Conference on Learning Representations* (2022).

[143] Hengyuan Hu and Dorsa Sadigh. “Language Instructed Reinforcement Learning for Human-AI Coordination”. In: *International Conference on Machine Learning (ICML)*. 2023.

[144] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. “Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents”. In: *arXiv preprint arXiv:2201.07207* (2022), pp. 9118–9147.

[145] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. "VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models". In: *Conference on Robot Learning*. PMLR. 2023, pp. 540–562.

[146] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. "Inner Monologue: Embodied Reasoning through Planning with Language Models". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1769–1782.

[147] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. "Inner Monologue: Embodied Reasoning through Planning with Language Models". In: *arXiv preprint arXiv:2207.05608*. 2022. arXiv: 2207.05608 [cs.RO].

[148] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. "Movement imitation with nonlinear dynamical systems in humanoid robots". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[149] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. "Rlbench: The robot learning benchmark & learning environment". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3019–3026.

[150] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *5th Annual Conference on Robot Learning*. 2021. URL: https://openreview.net/forum?id=8kbp23tSGYv.

[151] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *Conference on Robot Learning (CoRL)*. 2021.

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

[153] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. "Mistral 7B". In: *arXiv preprint arXiv:2401.04088* (2023). arXiv: 2310.06825 [cs.CL].

[154] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. "VIMA: General Robot Manipulation with Multimodal Prompts". In: *International Conference on Machine Learning*. 2023.

[155] Leslie Pack Kaelbling. "Learning to Achieve Goals". In: *IN PROC. OF IJCAI-93*. Morgan Kaufmann, 1993, pp. 1094–1098.

[156] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. 2018.

[157] Dmitry Kalashnkov, Jake Varley, Yevgen Chebotar, Ben Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. "MT-OPT: Continuous Multi-Task Robotic Reinforcement Learning at Scale". In: *arXiv* (2021).

[158] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. "CoTracker: It is Better to Track Together". In: *Proceedings ECCV*. 2024.

[159] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. "Cotracker: It is better to track together". In: *European Conference on Computer Vision*. Springer. 2025, pp. 18–35.

[160] Simar Kareer, Dhruv Patel, Ryan Punamiya, Pranay Mathur, Shuo Cheng, Chen Wang, Judy Hoffman, and Danfei Xu. *EgoMimic: Scaling Imitation Learning via Egocentric Video*. 2024. arXiv: 2410.24221 [cs.RO].

[161] Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. "Imitation Learning as f-Divergence Minimization". In: *arXiv preprint 1905.12888* (2020). arXiv: 1905.12888 [cs.LG].

[162] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. "3D Diffuser Actor: Policy Diffusion with 3D Scene Representations". In: *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*. 2024.

[163] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. "DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset". In: (2024).

[164] Changyeon Kim, Minho Heo, Doohyun Lee, Jinwoo Shin, Honglak Lee, Joseph J Lim, and Kimin Lee. “Subtask-Aware Visual Reward Learning from Segmented Demonstrations”. In: *International Conference on Learning Representations (ICLR)*. 2025.

[165] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. “OpenVLA: An Open-Source Vision-Language-Action Model”. In: *Conference on Robot Learning (CoRL)*. 2024.

[166] Moo Jin Kim, Jiajun Wu, and Chelsea Finn. “Giving Robots a Hand: Learning Generalizable Manipulation with Eye-in-Hand Human Video Demonstrations”. In: *CoRR* (2023).

[167] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *ICLR*. 2014.

[168] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. “Compositional Imitation Learning: Explaining and executing one task at a time”. In: *ICML* (2019).

[169] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[170] Yigit Korkmaz and Erdem Bıyık. “MILE: Model-based Intervention Learning”. In: *International Conference on Robotics and Automation (ICRA)*. 2025.

[171] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. “Offline Reinforcement Learning with Implicit Q-Learning”. In: *International Conference on Learning Representations (ICLR)*. 2022.

[172] Yuxuan Kuang, Junjie Ye, Haoran Geng, Jiageng Mao, Congyue Deng, Leonidas Guibas, He Wang, and Yue Wang. “Ram: Retrieval-based affordance transfer for generalizable zero-shot robotic manipulation”. In: *arXiv preprint arXiv:2407.04689* (2024).

[173] Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. “Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution”. In: *International Conference on Learning Representations*. 2022.

[174] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. “Stabilizing off-policy q-learning via bootstrapping error reduction”. In: *NeurIPS*. Vol. 32. 2019, pp. 11784–11794.

[175] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. “When should we prefer offline reinforcement learning over behavioral cloning?” In: *arXiv preprint arXiv:2204.05618* (2022).

[176] Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. “Maintaining Plasticity via Regenerative Regularization”. In: *arXiv preprint arXiv:2308.11958* (2023).

[177] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. “Reward Design with Language Models”. In: *International Conference on Learning Representations (ICLR)*. 2023.

[178] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. *CIC: Contrastive Intrinsic Control for Unsupervised Skill Discovery*. 2022. arXiv: 2202.00161 [cs.LG].

[179] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. "URLB: Unsupervised reinforcement learning benchmark". In: *NeurIPS* (2021). arXiv: 2110.15191 [cs.LG].

[180] Alex X. Lee, Coline Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, Claudio Fantacci, Jose Enrique Chen, Akhil Raju, Rae Jeong, Michael Neunert, Antoine Laurens, Stefano Saliceti, Federico Casarini, Martin Riedmiller, Raia Hadsell, and Francesco Nori. "Beyond Pick-and-Place: Tackling Robotic Stacking of Diverse Shapes". In: *Conference on Robot Learning (CoRL)*. 2021. URL: https://openreview.net/forum?id=U0Q8CrtBJxJ.

[181] Kimin Lee, Laura Smith, and Pieter Abbeel. "PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training". In: *International Conference on Machine Learning (ICML)*. 2021.

[182] Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. "Surgical fine-tuning improves adaptation to distribution shifts". In: *International Conference on Learning Representations* (2023).

[183] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. "Composing complex skills by learning transition policies". In: *ICLR*. 2018.

[184] Youngwoon Lee, Jingyun Yang, and Joseph J. Lim. "Learning to Coordinate Manipulation Skills via Skill Behavior Diversification". In: *ICLR*. Vol. 5. 47. American Association for the Advancement of Science, 2020, eabc5986. URL: https://openreview.net/forum?id=ryxB2lBtvH.

[185] Marion Lepert, Jiaying Fang, and Jeannette Bohg. *Phantom: Training Robots Without Robots Using Only Human Videos*. 2025. arXiv: 2503.00779 [cs.RO].

[186] Brian Lester, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning". In: *arXiv preprint arXiv:2104.08691* (2021).

[187] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020). arXiv: 2005.01643 [cs.LG].

[188] Haozhuo Li, Yuchen Cui, and Dorsa Sadigh. *How to Train Your Robots? The Impact of Demonstration Modality on Imitation Learning*. 2025.

[189] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. "Pre-Trained Language Models for Interactive Decision-Making". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. 2022. URL: https://openreview.net/forum?id=FWMQYjFso-a.

[190] Xiang Lisa Li and Percy Liang. “Prefix-tuning: Optimizing continuous prompts for generation”. In: *arXiv preprint arXiv:2101.00190* (2021).

[191] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. “Evaluating Real-World Robot Manipulation Policies in Simulation”. In: *arXiv preprint arXiv:2405.05941* (2024).

[192] Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. “HAMSTER: Hierarchical Action Models for Open-World Robot Manipulation”. In: *International Conference on Learning Representations (ICLR)*. 2025.

[193] Anthony Liang, Ishika Singh, Karl Pertsch, and Jesse Thomason. “Transformer Adapters for Robot Learning”. In: *CoRL 2022 Workshop on Pre-training Robot Learning*. 2022.

[194] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. “Code as policies: Language model programs for embodied control”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 9493–9500.

[195] William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. “Environment Curriculum Generation via Large Language Models”. In: *Conference on Robot Learning (CoRL)*. 2024.

[196] Haohong Lin, Radu Corcodel, and Ding Zhao. *Generalize by Touching: Tactile Ensemble Skill Transfer for Robotic Furniture Assembly*. 2024. arXiv: 2404.17684 [cs.RO].

[197] Li-Heng Lin, Yuchen Cui, Amber Xie, Tianyu Hua, and Dorsa Sadigh. “FlowRetrieval: Flow-Guided Data Retrieval for Few-Shot Imitation Learning”. In: *Conference on Robot Learning*. 2024.

[198] Ji Lin, Hongxu Yin, Wei Ping, Pavlo Molchanov, Mohammad Shoeybi, and Song Han. “VILA: On Pre-training for Visual Language Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 26689–26699.

[199] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. “Text2motion: From natural language instructions to feasible plans”. In: *Autonomous Robots* 47.8 (2023), pp. 1345–1365.

[200] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. “Deepseek-v3 technical report”. In: *arXiv preprint arXiv:2412.19437* (2024).

[201] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. *LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning*. 2023. arXiv: 2306.03310 [cs.AI].

[202] Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. “Moka: Open-vocabulary robotic manipulation through mark-based visual prompting”. In: *arXiv preprint arXiv:2403.03174* (2024).

[203] Guan-Ting Liu, En-Pei Hu, Pu-Jen Cheng, Hung-Yi Lee, and Shao-Hua Sun. “Hierarchical Programmatic Reinforcement Learning via Learning to Compose Programs”. In: *International Conference on Machine Learning*. 2023.

[204] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. “Improved baselines with visual instruction tuning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 26296–26306.

[205] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. “Visual instruction tuning”. In: *Advances in neural information processing systems* 36 (2024).

[206] Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. “Robot Learning on the Job: Human-in-the-Loop Autonomy and Learning During Deployment”. In: *Robotics: Science and Systems (RSS)*. 2023.

[207] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. “GPT understands, too”. In: *AI Open* (2023).

[208] Yao Liu, Pratik Chaudhari, and Rasool Fakoor. “Budgeting Counterfactual for Offline RL”. In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023, pp. 5729–5751.

[209] Zuxin Liu, Zijian Guo, Yihang Yao, Zhepeng Cen, Wenhao Yu, Tingnan Zhang, and Ding Zhao. “Constrained decision transformer for offline safe reinforcement learning”. In: *arXiv preprint arXiv:2302.07351* (2023).

[210] Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. “TAIL: Task-specific Adapters for Imitation Learning with Large Pretrained Models”. In: *International Conference on Learning Representations (ICLR)*. 2024.

[211] David Lopez-Paz and Marc’Aurelio Ranzato. “Gradient episodic memory for continual learning”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[212] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. “Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering”. In: *The 36th Conference on Neural Information Processing Systems (NeurIPS)*. 2022.

[213] Yao Lu, Karol Hausman, Yevgen Chebotar, Mengyuan Yan, Eric Jang, Alexander Herzog, Ted Xiao, Alex Irpan, Mohi Khansari, Dmitry Kalashnikov, and Sergey Levine. “AW-Opt: Learning Robotic Skills with Imitation and Reinforcement at Scale”. In: *CoRL*. 2021.

[214] Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. “Precise and Dexterous Robotic Manipulation via Human-in-the-Loop Reinforcement Learning”. In: *arXiv preprint arXiv:2410.21845*. 2024. arXiv: 2410.21845 [cs.RO].

[215] Clare Lyle, Mark Rowland, and Will Dabney. “Understanding and Preventing Capacity Loss in Reinforcement Learning”. In: *International Conference on Learning Representations*. 2022.

[216] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. “Learning Latent Plans from Play”. In: *Conference on Robot Learning*. 2020.

[217] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. “Learning latent plans from play”. In: *CoRL*. 2020, pp. 1113–1132.

[218] Corey Lynch and Pierre Sermanet. “Language Conditioned Imitation Learning over Unstructured Data”. In: *Robotics: Science and Systems* (2021). URL: https://arxiv.org/abs/2005.07648.

[219] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. “Interactive language: Talking to robots in real time”. In: *arXiv preprint arXiv:2210.06407* (2022). arXiv: 2210.06407 [cs.RO].

[220] Yecheng Jason Ma, Joey Hejna, Ayzaan Wahid, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Jonathan Tompson, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. “Vision Language Models are In-Context Value Learners”. In: *International Conference on Learning Representations (ICLR)*. 2025.

[221] Yecheng Jason Ma, William Liang, Vaidehi Som, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. “LIV: Language-Image Representations and Rewards for Robotic Control”. In: *International Conference on Machine Learning (ICML)*. 2023.

[222] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. “Eureka: Human-Level Reward Design via Coding Large Language Models”. In: *The Twelfth International Conference on Learning Representations*. 2024.

[223] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. “Eureka: Human-Level Reward Design via Coding Large Language Models”. In: *International Conference on Learning Representations (ICLR)*. 2024.

[224] Yecheng Jason Ma, William Liang, Hungju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. “DrEureka: Language Model Guided Sim-To-Real Transfer”. In: *Robotics: Science and Systems (RSS)*. 2024.

[225] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. “VIP: Towards Universal Visual Reward and Representation via Value-Implicit Pre-Training”. In: *The Eleventh International Conference on Learning Representations*. 2023.

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

[227] Arjun Majumdar, Karmesh Yadav, Sergio Arnaud, Yecheng Jason Ma, Claire Chen, Sneha Silwal, Aryan Jain, Vincent-Pierre Berges, Pieter Abbeel, Jitendra Malik, et al. “Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?” In: *arXiv preprint arXiv:2303.18240* (2023).

[228] Arjun Majumdar, Karmesh Yadav, Sergio Arnaud, Yecheng Jason Ma, Claire Chen, Sneha Silwal, Aryan Jain, Vincent-Pierre Berges, Pieter Abbeel, Jitendra Malik, Dhruv Batra, Yixin Lin, Oleksandr Maksymets, Aravind Rajeswaran, and Franziska Meier. "Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?" In: 2023. arXiv: 2303.18240 [cs.CV].

[229] Arun Mallya and Svetlana Lazebnik. "PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[230] Zhao Mandi, Homanga Bharadhwaj, Vincent Moens, Shuran Song, Aravind Rajeswaran, and Vikash Kumar. *CACTI: A Framework for Scalable Multi-Task Multi-Scene Visual Imitation Learning*. 2023. arXiv: 2212.05711 [cs.RO].

[231] Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. "MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1820–1864.

[232] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. *IRIS: Implicit Reinforcement without Interaction at Scale for Learning Control from Offline Robot Manipulation Data*. 2020. arXiv: 1911.05321 [cs.RO].

[233] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. "What Matters in Learning from Offline Human Demonstrations for Robot Manipulation". In: *arXiv preprint arXiv:2108.03298*. 2021.

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

[235] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. "UniPELT: A Unified Framework for Parameter-Efficient Language Model Tuning". In: 2022.

[236] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[237] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. "CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks". In: *IEEE Robotics and Automation Letters (RA-L)* 7.3 (2022), pp. 7327–7334.

[238] Marius Memmel, Jacob Berg, Bingqing Chen, Abhishek Gupta, and Jonathan Francis. "STRAP: Robot Sub-Trajectory Retrieval for Augmented Policy Learning". In: *International Conference on Learning Representations*. 2025.

[239] Russell Mendonca, Shikhar Bahl, and Deepak Pathak. "Structured World Models from Human Videos". In: *Conference on Robot Learning (CoRL)* (2023).

[240] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. *Discovering and Achieving Goals via World Models*. 2021.

[241] Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. "Catch & Carry: Reusable Neural Controllers for Vision-Guided Whole-Body Tasks". In: *ACM. Trans. Graph.* (2020).

[242] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. "Howto100m: Learning a text-video embedding by watching hundred million narrated video clips". In: *International Conference on Computer Vision (ICCV)*. 2019.

[243] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, et al. "Simple open-vocabulary object detection". In: *European Conference on Computer Vision*. Springer. 2022, pp. 728–755.

[244] Matthias Minderer, Alexey A. Gritsenko, and Neil Houlsby. "Scaling Open-Vocabulary Object Detection". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=mQPNcBWjGc.

[245] Suvir Mirchandani, Suneel Belkhale, Joey Hejna, Evelyn Choi, Md Sazzad Islam, and Dorsa Sadigh. "So You Think You Can Scale Up Autonomous Robot Data Collection?" In: *Conference on Robot Learning (CoRL)*. 2024.

[246] William Montgomery, Anurag Ajay, Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states". In: *International Conference on Robotics and Automation (ICRA)*. 2017.

[247] Meinard Müller. *Fundamentals of music processing: Using Python and Jupyter notebooks*. Vol. 2. Springer, 2021.

[248] Vivek Myers, Erdem Biyik, Nima Anari, and Dorsa Sadigh. "Learning Multimodal Rewards from Rankings". In: *Conference on Robot Learning (CoRL)*. 2021.

[249] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. "Data-efficient hierarchical reinforcement learning". In: *NeurIPS*. 2018.

[250] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. "Accelerating Online Reinforcement Learning with Offline Datasets". In: *arXiv preprint arXiv:2006.09359* (2020).

[251] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. *AWAC: Accelerating Online Reinforcement Learning with Offline Datasets*. 2021. arXiv: 2006.09359 [cs.LG].

[252] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. "R3M: A Universal Visual Representation for Robot Manipulation". In: *Conference on Robot Learning (CoRL)*. 2022.

[253] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. “R3M: A Universal Visual Representation for Robot Manipulation”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 892–909.

[254] Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. “Steering Your Generalists: Improving Robotic Foundation Models via Value Guidance”. In: *Conference on Robot Learning (CoRL)* (2024).

[255] Taewook Nam, Juyong Lee, Jesse Zhang, Sung Ju Hwang, Joseph J. Lim, and Karl Pertsch. “LiFT: Unsupervised Reinforcement Learning with Foundation Models as Teachers”. In: *arXiv preprint arXiv:2312.08958* (2023). arXiv: 2312.08958 [cs.LG].

[256] Taewook Nam, Shao-Hua Sun, Karl Pertsch, Sung Ju Hwang, and Joseph J. Lim. “Skill-based Meta-Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR)*. 2022.

[257] Taewook Nam, Shao-Hua Sun, Karl Pertsch, Sung Ju Hwang, and Joseph J. Lim. “Skill-based Meta-Reinforcement Learning”. In: *International Conference on Learning Representations*. 2022.

[258] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. “Learning and Retrieval from Prior Data for Skill-based Imitation Learning”. In: *Conference on Robot Learning*. 2022.

[259] Soroush Nasiriany, Sean Kirmani, Tianli Ding, Laura Smith, Yuke Zhu, Danny Driess, Dorsa Sadigh, and Ted Xiao. “RT-Affordance: Affordances are Versatile Intermediate Representations for Robot Manipulation”. In: *arXiv preprint arXiv:2411.02704* (Nov. 2024). URL: https://arxiv.org/abs/2411.02704.

[260] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. “Augmenting Reinforcement Learning with Behavior Primitives for Diverse Manipulation Tasks”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2022.

[261] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, et al. “PIVOT: Iterative Visual Prompting Elicits Actionable Knowledge for VLMs”. In: *International Conference on Machine Learning*. 2024.

[262] Andrew Y. Ng and Stuart J. Russell. “Algorithms for Inverse Reinforcement Learning”. In: *International Conference on Machine Learning (ICML)*. 2000.

[263] Nghia Nguyen, Minh Nhat Vu, Tung D Ta, Baoru Huang, Thieu Vo, Ngan Le, and Anh Nguyen. “Robotic-CLIP: Fine-tuning CLIP on Action Data for Robotic Applications”. In: *International Conference on Robotics and Automation (ICRA)*. 2025.

[264] Nghia Nguyen, Minh Nhat Vu, Tung D Ta, Baoru Huang, Thieu Vo, Ngan Le, and Anh Nguyen. “Robotic-CLIP: Fine-tuning CLIP on Action Data for Robotic Applications”. In: *ICRA*. 2025.

[265] Bolin Ni, Houwen Peng, Minghao Chen, Songyang Zhang, Gaofeng Meng, Jianlong Fu, Shiming Xiang, and Haibin Ling. “Expanding Language-Image Pretrained Models for General Video Recognition”. In: *European Conference on Computer Vision (ECCV)*. 2022.

[266] Dantong Niu, Yuvan Sharma, Giscard Biamby, Jerome Quenum, Yutong Bai, Baifeng Shi, Trevor Darrell, and Roei Herzig. "LLARVA: Vision-Action Instruction Tuning Enhances Robot Learning". In: *8th Annual Conference on Robot Learning*. 2024. URL: https://openreview.net/forum?id=Q2lGXMZCv8.

[267] Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. "Do Embodied Agents Dream of Pixelated Sheep?: Embodied Decision Making using Language Guided World Modelling". In: *International Conference on Machine Learning (ICML)*. 2023.

[268] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. "Zero-shot task generalization with multi-task deep reinforcement learning". In: *ICML*. 2017.

[269] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV].

[270] Georgios Papagiannis, Norman Di Palo, Pietro Vitiello, and Edward Johns. *R+X: Retrieval and Execution from Everyday Human Videos*. 2024. arXiv: 2407.12957 [cs.RO].

[271] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. "The unsurprising effectiveness of pre-trained vision models for control". In: *international conference on machine learning*. PMLR. 2022, pp. 17359–17371.

[272] Seohong Park, Kimin Lee, Youngwoon Lee, and Pieter Abbeel. *Controllability-Aware Unsupervised Skill Discovery*. 2023. arXiv: 2302.05103 [cs.RO].

[273] Alexander Pashevich, Cordelia Schmid, and Chen Sun. "Episodic Transformer for Vision-and-Language Navigation". In: *ICCV*. 2021.

[274] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. "Learning and generalization of motor skills by learning from demonstration". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 763–768.

[275] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. *Curiosity-driven Exploration by Self-supervised Prediction*. 2017. arXiv: 1705.05363 [cs.LG].

[276] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. *MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies*. 2019. arXiv: 1905.09808 [cs.LG]. URL: https://arxiv.org/abs/1905.09808.

[277] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning". In: *arXiv preprint arXiv:1910.00177*. 2019.

[278] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. *Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning*. 2019. arXiv: 1910.00177 [cs.LG].

[279] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. “Advantage-weighted regression: Simple and scalable off-policy reinforcement learning”. In: *arXiv preprint arXiv:1910.00177* (2019).

[280] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871 [cs.CV].

[281] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. “FiLM: Visual Reasoning with a General Conditioning Layer”. In: *AAAI*. 2018.

[282] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. “Accelerating Reinforcement Learning with Learned Skill Priors”. In: *Conference on Robot Learning (CoRL)*. 2020.

[283] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J. Lim. “Demonstration-Guided Reinforcement Learning with Learned Skills”. In: *CoRL*. 2021.

[284] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. “AdapterFusion: Non-destructive task composition for transfer learning”. In: *arXiv preprint arXiv:2005.00247* (2020).

[285] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. “Adapterhub: A framework for adapting transformers”. In: *arXiv preprint arXiv:2007.07779* (2020).

[286] Marc Pickett and Andrew G Barto. “Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning”. In: *ICML*. Vol. 19. ICML ’02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 506–513. ISBN: 1558608737.

[287] Dean A Pomerleau. “Alvinn: An autonomous land vehicle in a neural network”. In: *Advances in neural information processing systems* 1 (1988).

[288] Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. “The colosseum: A benchmark for evaluating generalization for robotic manipulation”. In: *arXiv preprint arXiv:2402.08191* (2024).

[289] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. *Learning Transferable Visual Models From Natural Language Supervision*. PMLR, 2021. arXiv: 2103.00020 [cs.CV].

[290] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language Models are Unsupervised Multitask Learners”. In: 2019.

[291] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language Models are Unsupervised Multitask Learners”. In: *OpenAI blog* 1.8 (2019), p. 9.

[292] Ilija Radosavovic, Baifeng Shi, Letian Fu, Ken Goldberg, Trevor Darrell, and Jitendra Malik. “Robot learning with sensorimotor pre-training”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 683–693.

[293] Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. *Scaling Language Models: Methods, Analysis & Insights from Training Gopher*. 2021. DOI: 10.48550/ARXIV.2112.11446.

[294] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. "Efficient off-policy meta-reinforcement learning via probabilistic context variables". In: *ICML*. 2019.

[295] Urs Ramer. "An iterative procedure for the polygonal approximation of plane curves". In: *Computer Graphics and Image Processing* 1.3 (1972), pp. 244–256. ISSN: 0146-664X. DOI: https://doi.org/10.1016/S0146-664X(72)80017-0.

[296] S. Rebuffi, A. Vedaldi, and H. Bilen. "Efficient Parametrization of Multi-domain Deep Neural Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2018, pp. 8119–8127. DOI: 10.1109/CVPR.2018.00847.

[297] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, et al. "A Generalist Agent". In: *Transactions on Machine Learning Research* (2022). Featured Certification, Outstanding Certification. ISSN: 2835-8856.

[298] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2019.

[299] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. "Vision-Language Models are Zero-Shot Reward Models for Reinforcement Learning". In: *NeurIPS 2023 Foundation Models for Decision Making Workshop*. 2023. URL: https://openreview.net/forum?id=JUwczEJY8I.

[300] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. "Vision-Language Models are Zero-Shot Reward Models for Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2024.

[301] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. "Experience replay for continual learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[302] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *AISTATS*. Vol. 15. Proceedings of Machine Learning Research. PMLR, Apr. 2011, pp. 627–635.

[303] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115 (2015), pp. 211–252.

[304] Dorsa Sadigh, Anca D. Dragan, S. Shankar Sastry, and Sanjit A. Seshia. "Active Preference-Based Learning of Reward Functions". In: *Robotics: Science and Systems (RSS)*. 2017.

[305] Stefan Schaal. "Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics". In: *Adaptive Motion of Animals and Machines*. Springer Tokyo, 2006.

[306] Stefan Schaal. "Dynamic Movement Primitives–A Framework for Motor Control in Humans and Humanoid Robotics". In: *Adaptive Motion of Animals and Machines* (Jan. 2006).

[307] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1312–1320. URL: https://proceedings.mlr.press/v37/schaul15.html.

[308] Jürgen Schmidhuber. "Learning Complex, Extended Sequences Using the Principle of History Compression". In: *Neural Computation* 4.2 (1992), pp. 234–242. DOI: 10.1162/neco.1992.4.2.234.

[309] Jürgen Schmidhuber, Jieyu Zhao, and Marco Wiering. "Shifting Inductive Bias with Success-Story Algorithm, Adaptive Levin Search, and Incremental Self-Improvement". In: *Machine Learning* 28.1 (July 1997), pp. 105–130. ISSN: 1573-0565.

[310] Thomas Schmied, Markus Hofmarcher, Fabian Paischer, Razvan Pascanu, and Sepp Hochreiter. "Learning to Modulate pre-trained Models in RL". In: *arXiv preprint arXiv:2306.14884* (2023).

[311] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[312] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. *Planning to Explore via Self-Supervised World Models*. 2020. arXiv: 2005.05960 [cs.LG].

[313] Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. "State entropy maximization with random encoders for efficient exploration". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9443–9454.

[314] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. "Behavior Transformers: Cloning k modes with one stone". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. 2022.

[315] Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. "Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action". In: *Conference on Robot Learning*. 2022. arXiv: TBD. URL: https://arxiv.org/abs/2207.04429.

[316] Rutav M Shah and Vikash Kumar. "RRL: Resnet as representation for Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9465–9476.

[317] Tanmay Shankar and Abhinav Gupta. "Learning robot skills with temporal variational inference". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8624–8633.

[318] Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. "Discovering Motor Programs by Recomposing Demonstrations". In: *ICLR*. 2019.

[319] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. "Dynamics-Aware Unsupervised Discovery of Skills". In: *arXiv* abs/1907.01657 (2019).

[320] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. "Dynamics-aware unsupervised discovery of skills". In: *ICLR* (2020).

[321] Archit Sharma, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. "Autonomous reinforcement learning via subgoal curricula". In: *Advances in Neural Information Processing Systems* (2021).

[322] Mohit Sharma, Claudio Fantacci, Yuxiang Zhou, Skanda Koppula, Nicolas Heess, Jon Scholz, and Yusuf Aytar. "Lossless Adaptation of Pretrained Vision Models for Robotic Manipulation". In: *arXiv preprint arXiv:2304.06600* (2023).

[323] Zejiang Shen, Kyle Lo, Lucy Lu Wang, Bailey Kuehl, Daniel S. Weld, and Doug Downey. "Incorporating Visual Layout Structures for Scientific Text Classification". In: *ArXiv* abs/2106.00676 (2021). URL: https://arxiv.org/abs/2106.00676.

[324] Lucy Xiaoyang Shi, Joseph J. Lim, and Youngwoon Lee. "Skill-based Model-based Reinforcement Learning". In: *Conference on Robot Learning*. 2022.

[325] Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. "Taco: Learning task decomposition via temporal alignment for control". In: *ICML* (2018).

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

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

[328] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. "ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. URL: https://arxiv.org/abs/1912.01734.

[329] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. "Parrot: Data-Driven Behavioral Priors for Reinforcement Learning". In: *ICLR* (2021).

[330] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. "COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. 2020.

[331] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. "Progprompt: Generating situated robot task plans using large language models". In: *ICRA*. 2023. arXiv: 2209.11302 [cs.RO].

[332] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. "Progprompt: Generating situated robot task plans using large language models". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11523–11530.

[333] Charlie Snell, Ilya Kostrikov, Yi Su, Mengjiao Yang, and Sergey Levine. *Offline RL for Natural Language Generation with Implicit Language Q Learning*. 2023. arXiv: 2206.11871 [cs.CL].

[334] Sumedh Anand Sontakke, Jesse Zhang, Séb Arnold, Karl Pertsch, Erdem Biyik, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. "RoboCLIP: One Demonstration is Enough to Learn Robot Policies". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=DVlawv2rSI.

[335] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. "REGENT: A Retrieval-Augmented Generalist Agent That Can Act In-Context in New Environments". In: *International Conference on Learning Representations*. 2025.

[336] Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Sean Kirmani, Brianna Zitkovich, Fei Xia, et al. "Open-World Object Manipulation using Pre-Trained Vision-Language Models". In: *Conference on Robot Learning*. PMLR. 2023, pp. 3397–3417.

[337] Shao-Hua Sun, Te-Lin Wu, and Joseph J. Lim. "Program Guided Agent". In: *ICLR*. 2020.

[338] Priya Sundaresan, Suneel Belkhale, Dorsa Sadigh, and Jeannette Bohg. "KITE: Keypoint-Conditioned Policies for Semantic Manipulation". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1006–1021.

[339] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[340] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[341] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[342] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112 (1999), pp. 181–211.

[343] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112.1 (1999), pp. 181–211. ISSN: 0004-3702.

[344] Matthew E Taylor and Peter Stone. "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.7 (2009).

[345] Gemini Team. "Gemini: A Family of Highly Capable Multimodal Models". In: *arXiv preprint arXiv:2312.11805* (2024). arXiv: 2312.11805 [cs.CL].

[346] Gemini Robotics Team et al. "Gemini Robotics: Bringing AI into the Physical World". In: *arXiv preprint arXiv:2503.20020* (2025). arXiv: 2503.20020 [cs.RO].

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

[348] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. "Reinforcement learning of motor skills in high dimensions: A path integral approach". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2397–2403.

[349] Sebastian Thrun. "Is learning the n-th thing any easier than learning the first?" In: *Advances in neural information processing systems*. 1996, pp. 640–646.

[350] Sebastian Thrun and Tom M Mitchell. "Lifelong robot learning". In: *The biology and technology of intelligent autonomous agents*. Springer, 1995, pp. 165–196.

[351] Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. "Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation". In: *Robotics: Science and Systems* (2024).

[352] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[353] René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, Natalia Díaz Rodríguez, and David Filliat. "DisCoRL: Continual Reinforcement Learning via Policy Distillation". In: *CoRR* abs/1907.05855 (2019). arXiv: 1907.05855.

[354] Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J. Lim. "Learning to Synthesize Programs as Interpretable and Generalizable Policies". In: *Neural Information Processing Systems*. 2021.

[355] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. *Jump-Start Reinforcement Learning*. 2023. arXiv: 2204.02372 [cs.LG].

[356] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. "Jump-Start Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2023.

[357] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep image prior". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9446–9454.

[358] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008.

[359] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[360] Sreyas Venkataraman, Yufei Wang, Ziyu Wang, Zackory Erickson, and David Held. "Real-World Offline Reinforcement Learning from Vision Language Model Feedback". In: *arXiv preprint arXiv:2411.05273*. 2024.

[361] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272.

[362] Homer Rich Walke, Kevin Black, Tony Z. Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, Abraham Lee, Kuan Fang, Chelsea Finn, and Sergey Levine. "BridgeData V2: A Dataset for Robot Learning at Scale". In: *Conference on Robot Learning (CoRL)*. 2023.

[363] Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. "Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021.

[364] Ben Wang and Aran Komatsuzaki. *GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model*. https://github.com/kingoflolz/mesh-transformer-jax. May 2021.

[365] Lirui Wang, Xinlei Chen, Jialiang Zhao, and Kaiming He. "Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers". In: *Neural Information Processing Systems* 37 (2024), pp. 124420–124450.

[366] Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. “Tracking everything everywhere all at once”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 19795–19806.

[367] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. “RL-VLM-F: Reinforcement Learning from Vision Language Foundation Model Feedback”. In: *International conference on machine learning*. 2024.

[368] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. “RL-VLM-F: Reinforcement Learning from Vision Language Foundation Model Feedback”. In: *International Conference on Machine Learning (ICML)*. 2024.

[369] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. “Rl-vlm-f: Reinforcement learning from vision language foundation model feedback”. In: *International Conference on Machine Learning*. 2024.

[370] David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. “Unsupervised Control Through Non-Parametric Discriminative Rewards”. In: *ICLR*. 2019.

[371] Chuan Wen, Xingyu Lin, John So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. “Any-point trajectory modeling for policy learning”. In: *arXiv preprint arXiv:2401.00025* (2023).

[372] Nils Wilde, Erdem Biyik, Dorsa Sadigh, and Stephen L. Smith. “Learning Reward Functions from Scale Feedback”. In: *Conference on Robot Learning (CoRL)*. 2021.

[373] Jeffrey Wu, Seohong Park, Zipeng Lin, Jianlan Luo, and Sergey Levine. *V-Former: Offline RL with Temporally-Extended Actions*. 2024.

[374] Ted Xiao, Harris Chan, Pierre Sermanet, Ayzaan Wahid, Anthony Brohan, Karol Hausman, Sergey Levine, and Jonathan Tompson. “Robotic Skill Acquistion via Instruction Augmentation with Vision-Language Models”. In: *Proceedings of Robotics: Science and Systems*. 2023.

[375] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. “Masked Visual Pre-training for Motor Control”. In: *arXiv preprint arXiv:2203.06173* (2022).

[376] Jianan Xie, Zhen Xu, Jiayu Zeng, Yuyang Gao, and Kenji Hashimoto. “Human–Robot Interaction Using Dynamic Hand Gesture for Teleoperation of Quadruped Robots with a Robotic Arm”. In: *Electronics* (2025).

[377] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. “Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification”. In: *European Conference on Computer Vision (ECCV)*. 2018.

[378] Haoyu Xiong, Haoyuan Fu, Jieyi Zhang, Chen Bao, Qiang Zhang, Yongxi Huang, Wenqiang Xu, Animesh Garg, and Cewu Lu. “Robotube: Learning household manipulation from human videos with simulated twin environments”. In: *Conference on Robot Learning*. PMLR. 2023.

[379] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. "Neural task programming: Learning to generalize across hierarchical tasks". In: *ICRA*. IEEE. 2018.

[380] Haofei Xu, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, and Dacheng Tao. "Gmflow: Learning optical flow via global matching". In: *Conference on Computer Vision and Pattern Recognition*. 2022.

[381] Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. "Flow as the Cross-domain Manipulation Interface". In: *8th Annual Conference on Robot Learning*. 2024.

[382] Mengdi Xu, Yuchen Lu, Yikang Shen, Shun Zhang, Ding Zhao, and Chuang Gan. "Hyper-decision transformer for efficient online policy adaptation". In: *arXiv preprint arXiv:2304.08487* (2023).

[383] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. "Prompting decision transformer for few-shot policy generalization". In: *international conference on machine learning*. PMLR. 2022, pp. 24631–24645.

[384] Ge Yan, Kris Wu, and Xiaolong Wang. *UCSD kitchens Dataset*. Aug. 2023.

[385] Brian Yang, Jesse Zhang, Vitchyr Pong, Sergey Levine, and Dinesh Jayaraman. "REPLAB: A Reproducible Low-Cost Arm Benchmark for Robotic Learning". In: *ICRA*. 2019, pp. 8691–8697. DOI: 10.1109/ICRA.2019.8794390.

[386] Daniel Yang, Davin Tjia, Jacob Berg, Dima Damen, Pulkit Agrawal, and Abhishek Gupta. "Rank2Reward: Learning Shaped Reward Functions from Passive Video". In: *International Conference on Robotics and Automation (ICRA)*. 2024.

[387] Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. "Robot Fine-Tuning Made Easy: Pre-Training Rewards and Policies for Autonomous Real-World Reinforcement Learning". In: *International Conference on Robotics and Automation (ICRA)*. 2024.

[388] Zhaojing Yang, Miru Jun, Jeremy Tien, Stuart J. Russell, Anca Dragan, and Erdem Bıyık. "Trajectory Improvement and Reward Learning from Comparative Language Feedback". In: *Conference on Robot Learning (CoRL)*. 2024.

[389] Yihang Yao, Zuxin Liu, Zhepeng Cen, Jiacheng Zhu, Wenhao Yu, Tingnan Zhang, and Ding Zhao. "Constraint-conditioned policy optimization for versatile safe reinforcement learning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[390] Weirui Ye, Yunsheng Zhang, Haoyang Weng, Xianfan Gu, Shengjie Wang, Tong Zhang, Mengchen Wang, Pieter Abbeel, and Yang Gao. "Reinforcement Learning with Foundation Priors: Let Embodied Agent Efficiently Learn on Its Own". In: *Conference on Robot Learning (CoRL)*. 2024.

[391] Sarah Young, Jyothish Pari, Pieter Abbeel, and Lerrel Pinto. "Playful interactions for representation learning". In: *International Conference on Intelligent Robots and Systems*. IEEE. 2022.

[392] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. *Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning*. 2019. arXiv: 1910.10897 [cs.LG].

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

[394] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. “Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning”. In: *Conference on Robot Learning (CoRL)*. 2019.

[395] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Dee M, Jodilyn Peralta, Brian Ichter, Karol Hausman, and Fei Xia. “Scaling Robot Learning with Semantically Imagined Experience”. In: *arXiv preprint arXiv:2302.11550*. 2023.

[396] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. “Language to Rewards for Robotic Skill Synthesis”. In: *Conference on Robot Learning (CoRL)*. 2023.

[397] Chengbo Yuan, Chuan Wen, Tong Zhang, and Yang Gao. “General flow as foundation affordance for scalable robot learning”. In: *arXiv preprint arXiv:2401.11439* (2024).

[398] Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. “RoboPoint: A Vision-Language Model for Spatial Affordance Prediction in Robotics”. In: *8th Annual Conference on Robot Learning*. 2024. URL: https://openreview.net/forum?id=GVX6jpZOhU.

[399] Grace Zhang, Ayush Jain, Injune Hwang, Shao-Hua Sun, and Joseph J. Lim. *QMP: Q-switch Mixture of Policies for Multi-Task Behavior Sharing*. 2023. arXiv: 2302.00671 [cs.LG].

[400] Grace Zhang, Linghan Zhong, Youngwoon Lee, and Joseph J Lim. “Policy Transfer across Visual and Dynamics Domain Gaps via Iterative Grounding”. In: *RSS*. 2021.

[401] Jesse Zhang, Brian Cheung, Chelsea Finn, Sergey Levine, and Dinesh Jayaraman. “Cautious Adaptation For Reinforcement Learning in Safety-Critical Settings”. In: *ICML*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 11055–11065. URL: https://proceedings.mlr.press/v119/zhang20e.html.

[402] Jesse Zhang, Minho Heo, Zuxin Liu, Erdem Biyik, Joseph J Lim, Yao Liu, and Rasool Fakoor. “EXTRACT: Efficient Policy Learning by Extracting Transferrable Robot Skills from Offline Data”. In: *Conference on Robot Learning (CoRL)*. 2024.

[403] Jesse Zhang, Karl Pertsch, Jiefan Yang, and Joseph J Lim. “Minimum Description Length Skills for Accelerated Reinforcement Learning”. In: *Self-Supervision for Reinforcement Learning Workshop - ICLR 2021*. 2021.

[404] Jesse Zhang, Karl Pertsch, Jiahui Zhang, and Joseph J. Lim. “SPRINT: Scalable Policy Pre-Training via Language Instruction Relabeling”. In: *arXiv preprint arXiv:2306.11886* (2023). arXiv: 2306.11886 [cs.RO].

[405] Jesse Zhang, Karl Pertsch, Jiahui Zhang, and Joseph J. Lim. "SPRINT: Scalable Policy Pre-Training via Language Instruction Relabeling". In: *International Conference on Robotics and Automation (ICRA)*. 2024.

[406] Jesse Zhang, Haonan Yu, and Wei Xu. "Hierarchical Reinforcement Learning by Discovering Intrinsic Options". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=r-gPPHEjpmw.

[407] Jesse Zhang, Jiahui Zhang, Karl Pertsch, Ziyi Liu, Xiang Ren, Minsuk Chang, Shao-Hua Sun, and Joseph J Lim. "Bootstrap Your Own Skills: Learning to Solve New Tasks with Large Language Model Guidance". In: *Conference on Robot Learning (CoRL)*. 2023.

[408] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. *OPT: Open Pre-trained Transformer Language Models*. 2022. DOI: 10.48550/ARXIV.2205.01068.

[409] Zichen Zhang, Yunshuang Li, Osbert Bastani, Abhishek Gupta, Dinesh Jayaraman, Yecheng Jason Ma, and Luca Weihs. *Universal Visual Decomposer: Long-Horizon Manipulation Made Easy*. 2023. arXiv: 2310.08581 [cs.RO]. URL: https://arxiv.org/abs/2310.08581.

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

[411] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware". In: *Robotics: Science and Systems*. Ed. by Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu. 2023.

[412] Tony Z. Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. *ALOHA Unleashed: A Simple Recipe for Robot Dexterity*. 2024. arXiv: 2410.13126 [cs.RO].

[413] Qinqing Zheng, Amy Zhang, and Aditya Grover. *Online Decision Transformer*. 2022. arXiv: 2202.05607 [cs.LG].

[414] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. *On the Continuity of Rotation Representations in Neural Networks*. 2020. arXiv: 1812.07035 [cs.LG]. URL: https://arxiv.org/abs/1812.07035.

[415] Zhiyuan Zhou, Pranav Atreya, Abraham Lee, Homer Rich Walke, Oier Mees, and Sergey Levine. "Autonomous Improvement of Instruction Following Skills via Foundation Models". In: *Conference on Robot Learning (CoRL)*. 2024.

[416] Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. "Efficient Online Reinforcement Learning Fine-Tuning Need Not Retain Offline Data". In: *International Conference on Learning Representations (ICLR)*. 2025.

[417] Wanrong Zhu, Jack Hessel, Anas Awadalla, Samir Yitzhak Gadre, Jesse Dodge, Alex Fang, Youngjae Yu, Ludwig Schmidt, William Yang Wang, and Yejin Choi. "Multimodal C4: An Open, Billion-scale Corpus of Images Interleaved With Text". In: *arXiv preprint arXiv:2304.06939* (2023).

[418] Xinghao Zhu, Ran Tian, Chenfeng Xu, Mingyu Ding, Wei Zhan, and Masayoshi Tomizuka. *Fanuc Manipulation: A Dataset for Learning-based Manipulation with FANUC Mate 200iD Robot.* https://sites.google.com/berkeley.edu/fanuc-manipulation. 2023.

[419] Yifeng Zhu, Peter Stone, and Yuke Zhu. "Bottom-Up Skill Discovery From Unsegmented Demonstrations for Long-Horizon Robot Manipulation". In: *IEEE Robotics and Automation Letters (RA-L)* (2022).

[420] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. "robosuite: A modular simulation framework and benchmark for robot learning". In: *arXiv preprint arXiv:2009.12293* (2020).

[421] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. "Maximum Entropy Inverse Reinforcement Learning". In: *AAAI Conference on Artificial Intelligence*. 2008.

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

Appendices

Appendix A

SPRINT

A.1 Large Language Model Prompt

We list the full large language model summarization prompt in Figure A.1. The examples in the prompt are fixed for all summarization queries. These examples are selected from the ALFRED validation dataset (which is not otherwise used in our work) at random: We spell out the primitive skill annotations in the "Task Steps:" part of each prompt example. Then, the "Summary" for each of these is the high-level, human-written annotation for that trajectory from ALFRED. We repeatedly sampled these trajectories until each example mentioned a different object to prevent biasing the LLM towards certain types of objects.

We note that the "Look at the box under the lamp light" example is important to make the LLM give reasonable summaries for similar tasks in ALFRED where the agent picks something up and turns on a light. This is because most of the human labels for turning on the lamp do not mention the object in the previous step, making it difficult for the LLM to realize that the task has to do with looking at the held object under a lamp.

A.2 Baselines and Implementation

We implement IQL [171] as the base offline RL algorithm for all goal-conditioned offline RL pretraining baselines and ablations due to its strong offline and finetuning performance on a variety of dense and sparse reward environments. At a high level, IQL trains on in-distribution (s, a, s', r, a') tuples from the dataset rather than sampling a policy for a' to ensure that the Q and value functions represent accurate estimated returns constrained to actions in the dataset. The value function is trained with an expectile regression loss controlled by a hyperparameter τ, where $\tau = 0.5$ results in standard mean squared error loss and $\tau \to 1$ approximates the max operator, resulting in a more optimistic value function that can better "stitch" together trajectories to obtain distant reward in sparse reward settings. The IQL policy is trained to maximize the following objective:

$$e^{\beta(Q(s,a)-V(s))} \log \pi(a|s),$$

which performs advantage-weighted regression [277] with an inverse temperature term β. In practice, the exponential advantage term is limited to a maximum value to avoid numerical overflow issues. We detail shared training and implementation details below, with method-specific information and hyperparameters in the following subsections.

A.2.1 ALFRED Details

Observation space. The state space of the ALFRED environment consists of 300×300 RGB images. Following the baseline method in ALFRED [328], we preprocess these images by sending them through a frozen ResNet-18 encoder [133] pretrained on ImageNet [78]. This results in a $512 \times 7 \times 7$ feature map that we use as the observation input to all networks.

Action space. The agent chooses from 12 discrete low-level actions. There are 5 navigation actions: `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, and `LookDown` and 7 interaction actions: `Pickup`, `Put`, `Open`, `Close`, `ToggleOn`, `ToggleOff`, and `Slice`. For interaction actions the agent additionally selects one of 82 object types to interact with, as defined by Pashevich, Schmid, and Sun [273]. In total, the action space consists of $5 + 7 * 82 = 579$ discrete action choices. Note that this action space definition is different from the action space in Shridhar et al. [328], which used a pixel-wise mask output to determine the object to interact with. In contrast to Shridhar et al. [328] we aim to train agents with *reinforcement learning* instead of imitation learning and found the discrete action parametrization more amenable to RL training than dense mask outputs. For all methods, due to the large discrete action space, we perform some basic action masking to prevent agents from taking actions that are not possible. For example, we do not allow the agent to `Close` objects that aren't closeable nor can they `ToggleOn` objects that can't be turned on.

Policy and critic networks. For all baselines and SPRINT base models are implemented on the transformer architecture proposed in Episodic Transformers [273]. For offline RL methods (AM, SPRINT) we follow the advice of [333] and parameterize both Q functions and the Value function of IQL as separate output heads of one transformer backbone that is used for all critic networks. We train both policies and critic transformer networks with an observation history of up to 16 previous observations, each one being processed by a convolutional network before being flattened into a 768-dim feature. Our discrete policy has two output heads of size 12 and 82 for the action and interaction object outputs respectively. Critic networks are conditioned on both the observation and the discrete action output of the policy. In networks with language input, words are individually tokenized and the entire language instruction is fed to the policy and critic networks and embedded into a sequence of learned 768-dim embeddings, one for each token. We perform cross-attention between all network inputs: language embeddings, previous observation embeddings, and the previous action where applicable. The output of this cross-attention mechanism is then transformed by linear layers into the final output for the network.

Pre-training hyperparameters. A hyperparameter search was performed first on the language-conditioned BC-baseline to optimize for training accuracy. These hyperparameters were carried over to the IQL implementation, and another search for IQL-specific hyperpameters were performed on a baseline IQL policy conditioned on language instructions. With these parameters fixed, we performed one more hyperparameter search specific to Actionable Models but for the final implementation of SPRINT we re-used the same hyperparameters and only selected SPRINT-specific parameters heuristically. Hyperparameters for each method are detailed in separate tables. Shared hyperparameters for all methods (where applicable) are listed below:

Param	Value
Batch Size	1024
# Training Batches	140k
Learning Rate	1e-4
Optimizer	AdamW
Dropout Rate	0.1
Weight Decay	0.1
Discount γ	0.97
Q Update Polyak Averaging Coefficient	0.005
Policy and Q Update Period	1/train iter
Nonlinearity	ReLU
IQL Advantage Clipping	[0, 100]
IQL Advantage Inverse Temperature β	5
IQL Quantile τ	0.8
Maximum Transformer Context Length	16

Finetuning details and hyperparameters. We fine-tune by running IQL on online-collected data without any of the chaining or aggregation steps. For all models, we finetune by sampling old pre-training data and newly collected data at a ratio of 70%/30%. Without this mixed batch training, we found the transformer-based networks to overfit to the new data, something we did not see when experimenting with standard MLPs. The newly collected data is also sampled from two separate buffers at equal proportions, one which contains trajectories that received at least 1 reward (i.e., completed one sub-task) and one that contains trajectories that received none. This is another transformer-specific adaptation we had to make for the models to train stably with IQL on online-collected data.

Each method is finetuned on every task in the $EVAL_{SCENE}$ task set individually; that is, we pre-train once and then finetune policies for each task in the task set. We then average returns over all tasks, then report metrics averaged over all random seeds. For each task, we define a maximum rollout time horizon of 2 timesteps per environment action required by an expert ALFRED task planner.

When not specified, finetuning parameters are identical to pre-training parameters. Finetuning hyperparameters are specified below:

Param	Value
# Initial Rollouts	50
Training to Env Step Ratio	20
ϵ in ϵ-greedy action sampling	0
Policy action sampling	True
# Parallel Rollout Samplers	10

A.2.2 Real Robot Implementation Details

The real-world environment uses a Kinova Jaco 2 robot arm. Below we detail the implementation and training details specific to the real robot environment.

Observation space. The view observations consist of 224×224×3 cropped RGB images, which are captured from a Logitech Pro Webcam C920 for the third-person view and an Intel RealSense D435 for the wrist-view. We leverage a pretrained R3M [252] model to encode each view observation. Additionally, the state representation includes the robot's end-effector position, velocity, and gripper state. Notably, the end-effector position and velocity are two continuous vectors, while the gripper state is represented as a one-hot vector, indicating OPEN, CLOSE, or NOT MOVE. To form the observation for the policy, we concatenate the embedded RGB input with state information.

To condition on language inputs, we use a pre-trained sentence embedder to embed the entire language annotation into a vector of size 384 (as our network backbone is an RNN instead of a transformer). This embedding is done with the `all-MiniLM-L12-v2` pre-trained embedding model from the `SentenceTransformers` package [298].

The total state input dimension is: 2048 (third-person R3M) + 2048 (wrist R3M) + 15 (Jaco state input) + 384 (language embedding) = 4495.

Action space. The robot action space comprises the changes in the end effector position between each time stamp, along with the gripper opening/closing commands. These actions are transmitted to the robot at a frequency of 10 Hz and interpreted as desired joint poses using PyBullet's inverse kinematics module.

Network architecture and training. Similar to [410], we use the Action Chunking method to train an autoregressive policy. Specifically, our policy employs an LSTM model to predict the next 15 actions, given the initial observation as input, i.e., $\pi(a_{t:t+15}|s_t)$. Our Q and Value networks are also recurrent, predicting per-timestep rewards for each action in the sequence. Just like the policy, they also only see the observation before the action sequence starts.

Because of the fact that the gripper action is discrete and heavily imbalanced in class distribution, we weigh the gripper action loss inversely proportionally to the number of examples in each class.

Pre-training details and hyperparameters. We performed a heuristic hyperparameter search by first tuning the language-conditioned BC baseline to be as effective as possible on zero-shot evaluations of training tasks, then performed a small heuristic hyperparameter for the SPRINT. Shared hyperparameters are detailed below:

Param	Value
Batch Size	128
# Training Batches	50k
Learning Rate	5e-4
Optimizer	AdamW
Weight Decay	0.1
Discount γ	0.99
Q Update Polyak Averaging Coefficient	0.005
Policy and Q Update Period	1/train iter
Nonlinearity	LeakyReLU(0.2)
IQL Advantage Clipping	[0, 100]
IQL Advantage Inverse Temperature β	5
IQL Quantile τ	0.8
Action Chunking Length	15

Fine-tuning details and hyperparameters. We collect 25 demonstrations for each downstream task and perform individual fine-tuning of the models for each task. In the case of the pre-trained models (SPRINT, L-BC composite, and L-BC primitive), we conduct 500 epochs of fine-tuning. As for the model without pre-training, we train 2000 epochs only on the downstream task demonstrations. The fine-tuning/training hyperparameters are identical to those for pre-training.

A.2.3 Language-conditioned Behavior Cloning

Our language-conditioned behavior cloning (L-BC) comparison method is inspired by and replicates BC-Zero [150] and LangLfP [218]. BC-Zero performs language imitation learning [281], and both BC-Zero and LangLfP have an additional image/video-language alignment objective. In BC-Zero, their video alignment objective aligns language embeddings with videos of humans performing tasks related to those the BC-Zero robot agent trains on. LangLfP's image-language alignment objective allows their policy to accept both image and natural language goals as input due to only having a subset of their data labeled with hindsight language labels. As we don't have human videos of these tasks and our entire dataset is labeled with language labels, we do not add a video or image alignment objective.

Hyperparameters for the L-BC baseline are identical to the shared parameters above for both environments, where applicable.

ALFRED: We implement L-BC by using the same architecture as described in the shared details section above with just a single transformer policy network that trains to maximize the log-likelihood of actions in the dataset. As our entire dataset consists of expert trajectories, this baseline ideally learns optimal actions for the instructions.

Real Robot: L-BC is implemented with the action-chunked LSTM policy network to maximize log-likelihood of actions in the dataset as described in the real robot implementation details section above. Again the dataset consists of human expert trajectories so L-BC should learn optimal actions for the given instructions.

A.2.4 Episodic Transformers

Episodic Transformers (ET) [273] trains a transformer architecture on full sequences of ALFRED instructions with a behavior cloning objective. This is currently state of the art in the "Seen Path-Length Weighted

Success Rate" evaluation metric on the ALFRED leaderboard. We adopted the ET implementation from the official code repository.

For fair comparison, we make a few modifications to make it as close as possible to SPRINT and the baselines: 1) we train it on the same dataset as all baselines, so we do not generate new synthetic training data like the original implementation Pashevich, Schmid, and Sun [273] since it assumes access to an expert planner, 2) we encode visual frames with a Resnet-18 instead of Resnet-50 backbone, the same we use for all other models, 3) we remove the high-level goal specification from the input text tokens as we do not assume access to those, and 4) we train the model for longer to match the number of training steps for all methods.

A.2.5 Actionable Models (AM)

Actionable Models [49] pre-trains a goal-conditioned Q function conditioned on randomly sampled image goals and also performs a goal-chaining procedure very similar to our skill chaining procedure. We implement AM by modifying the base IQL policy and critic networks to take in image goals instead of natural language embeddings as goals. These goals are provided in the same way as the observations as a sequence of 5 frames (the last 5 frames in the trajectory) processed by a frozen ResNet-18.

To allow for fair comparison between our approach and AM, we implement AM with the same powerful offline RL algorithm, IQL [171], used in our method. IQL ensures that the policy does not choose out-of-distribution actions by using advantage-weighted regression on in-distribution actions for policy extraction. With this, we found the conservative auxiliary loss AM adds to push down Q-values for out-of-distribution actions to be unnecessary and even hurtful to its overall performance, so we omit this additional loss term.

We also pre-train AM on the same long-horizon trajectories as those generated by SPRINT during LLM-based skill aggregation. This ensures a fair comparison in terms of the types and lengths of tasks seen during pre-training.

Finally, after consulting the authors of AM, we tried varying maximum trajectory lengths when sampling random goals. We found that allowing random goals to be sampled from anywhere within a trajectory resulted in the best zero-shot evaluation performance for AM, so our numbers are reported with this implementation detail.

A.2.6 SPRINT

The implementation details of SPRINT follow from the general discussions at the top of this section. The key differences are in (1) language model skill aggregation and (2) cross-trajectory skill chaining, detailed below.

LLM Skill Aggregation. We perform LLM skill aggregation fully offline by iterating through every trajectory and aggregating sequences of adjacent primitive skill sub-trajectories. Assuming a trajectory with N primitive skills, we select all $\binom{N}{2}$ pairs of start and end skills and aggregate all instructions from start to end with the LLM. With 73k original language-annotated sub-trajectories in ALFRED, this procedure allows us to generate an additional 110k aggregated trajectories. We then add these trajectories to the original dataset and train on the entire set.

On our real-world robot dataset, we start with $\sim$6k language-annotated sub-trajectories and perform LLM skill aggregation on all pairs of trajectories directly next to each other (restricting to a maximum of 2 skills being aggregated at any time). We restrict aggregation in this manner because each trajectory contains many sub-trajectories of play-like data where many of the sub-trajectories are not related to each other. Aggregation doubles the size of our dataset to almost $\sim$13k trajectories.

Cross-trajectory skill chaining. We perform cross-trajectory skill chaining in-batch. Instead of sampling a second trajectory to perform chaining on, we simply permute the batch indicies to generate a set of randomly sampled second trajectories. Then, we perform a second loss function update, in addition to the original update on the sampled trajectories, with equal loss weighting, to apply the skill-chaining update. We apply the chaining procedures from Eq. 3.3 in-batch. Empirically, we found that cross-trajectory skill chaining works slightly better with the on-policy Value function obtained through IQL, therefore we use state values at the chaining targets instead of state-action Q-values.

SPRINT-specific hyperparameters follow:

Param	Value
LLM	LLAMA-13B [352]
LLM Token Filtering Top-p	0.9
LLM Token Sampling Temperature	0.6

A.2.6.1 Cross-trajectory chaining preserves the MDP.

When performing cross-trajectory chaining using Eq. 3.3, special care must be taken to preserve the dynamics of the original Markov Decision Process (MDP). When chaining together two trajectories τ_A and τ_B, we concatenate the two sentences of each trajectory together and relabel their rewards with Eq. 3.3. The new language annotation used to chain together these trajectories is the concatenation of the two sentences, implying that the agent finishes skill (A) and then skill (B). However, we cannot concatenate the two trajectories together into one longer trajectory, as doing so would imply that the agent can instantaneously jump from the last state of skill (A) to the first state of skill (B), which may not be possible. Therefore, we instead treat the relabeled trajectories as separate trajectories with the same language annotation (lines 36 and 37 of Algorithm 1).

However, this introduces two possible complications: 1) Language annotations differing in structure from those in the original dataset, and 2) Possible instruction ambiguity. We detail how these complications are resolved in SPRINT below:

1. **Language annotations differing in structure.** Language annotations produced by the chaining procedure will result in annotations that implicitly skip certain steps. For example, when chaining skill (A), "make the bed," and skill (B), "make a cup of coffee," the resulting chained annotation will be "Make the bed. Make a cup of coffee." However to perform skill (B) the agent needs to first move to the kitchen from the bedroom to make the cup of coffee, which is skipped in this annotation. LLM-based skill aggregation (Section 3.3.2) helps bridge this gap by summarizing long-horizon sequences while skipping certain implied steps. For example, one real LLM summary summarized the sequence: "*1: Pick up the plaid pillow that is on the left end of the couch. 2: Place the pillow on the ottoman*" into the instruction "*Place a plaid pillow on the ottoman,*", which skipped the step of picking up the pillow as it is implied that you must do so before placing the pillow down. Using the LLM augments our original dataset such that, in ALFRED, we have 2.5x the original data after performing offline skill aggregation, and in the real robot manipulation environment we have 2x the amount of original data. Therefore after performing LLM aggregation, there are many examples of similar instructions to those used for chained trajectories that imply certain steps without mentioning them explicitly.

2. **Instruction ambiguity.** When chaining trajectories, there will be some ambiguity introduced as we do not have intermediate instructions for going from the last state of A to the initial state of B (obtaining these instructions requires additional human effort). This ambiguity is only present in the states of trajectory A, as when training on trajectory B, the agent can easily infer that the instructions

for trajectory A are finished and the just follow the instructions relevant for trajectory B. We believe that the effects of the ambiguity on pre-training performance depends greatly on the given dataset. In complex and diverse environments, hindsight-labeled annotations should contain details specific to certain scenes, resolving this ambiguity. In ALFRED, the annotations usually contain information about the specific objects that the agent must interact with or locations that the agent must go to. For example, annotations for rinsing mugs typically are of the form "*clean the MUG in the sink,*", or annotations for picking up a candle will often say something like "*pick up the YELLOW CANDLE on the COUNTER,*", highlighting specific details regarding what the agent is supposed to do to complete the trajectory.

A.2.7 SayCan

In ALFRED experiments, we evaluate the performance of SayCan [7], a top-down LLM-planning approach that breaks down a high-level task into a sequence of steps that a language-conditioned policy can execute. SayCan does not perform any fine-tuning as it is not a pre-training method, instead we implement it by prompting a large language model to produce a probability distribution over the set of primitive skill instructions relevant for each task. Therefore it receives some privileged information over all of the other compared methods, including SPRINT, about which primitive skills to choose from in each evaluation task.

Specifically, we use LLaMA-13B directly at test time to produce plans, the same model that we used to perform LLM skill relabeling for SPRINT. The pre-trained policies are pre-trained on the same data as L-BC except that we also pre-train a value function to use with SayCan as it weighs skill predictions using both a pre-trained language-conditioned value function and the LLM-produced probabilities.

The prompt for SayCan, inspired by the prompt recommended in the original paper, and with the same number of examples as the one for SPRINT, follows below:

Robot: Hi there, I'm a robot operating in a house. Robot: You can ask me to do various tasks and I'll tell you the sequence of actions I would do to accomplish your task.

Human: How would you put the box with keys on the sofa?

Robot: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.

Human: How would you put a cooled slice of lettuce on the counter?

Robot: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.

Human: How would you put a book on the couch?

Robot: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.

Human: How would you put the cleaned fork in a drawer?

Robot: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.

Human: How would you put the box of tissues on the barred rack?

Robot: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.

Human: How would you put a heated glass on the wooden rack?

Robot: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.

Human: How would you look at the box under the lamp light?

Robot: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.

Human: How would you [HIGH LEVEL TASK DESCRIPTION]?

Robot: 1. [SKILL 1 EXECUTED SO FAR] 2. [SKILL 2 EXECUTED SO FAR] ... N. ____

A.3 Dataset, Environment, and Task Details

A.3.1 ALFRED

A.3.1.1 Dataset Details

For training and evaluation we leverage the ALFRED benchmark and dataset [328]. The ALFRED training dataset contains ∼6.6k trajectories collected by an optimal planner following a set of 7 high-level tasks with randomly sampled objects (e.g., pick up an object and heat it). Each trajectory has at least three crowd-sourced sets of language instruction annotations. Each trajectory consists of a sequence of 3-19 individually annotated skills (see Figure A.3, left). This results in a total of 141k language-annotated skill trajectories.

However, nearly half of the language instructions in the ALFRED dataset are navigation skill instructions like "turn left, then look up and walk to the counter on the right". To get a more balanced skill annotation dataset, we merge all navigation skills with the skill that immediately follows them, using only the annotation of the next skill. After this processing step, the resulting dataset contains 73k language-annotated primitive skill trajectories. After we merge the navigation skills, the average number of skills in each trajectory is 3.5 skills per trajectory (Figure A.3, middle), and the average number of actions in each skill is 14.3 (Figure A.3, right).

Table A.1: Evaluation Task Specifics. Note that the "number of env actions per task" corresponds to the number of environment actions the ALFRED expert planner required to complete that task.

	$EVAL_{INSTRUCT}$	$EVAL_{LENGTH}$	$EVAL_{SCENE}$
Number of Tasks	100	20	10
Task Lengths (# primitive skills)	[1, 2, 3, 4, 5, 6, 7]	[7, 8]	[1, 2, 3, 4, 5]
Min Number of Env Actions per Task	1	34	2
Avg Number of Env Actions per Task	39.1	60.9	46.6
Max Number of Env Actions per Task	113	104	124

A.3.1.2 Evaluation Tasks

Overview. We evaluate agents through zero-shot policy evaluation and finetuning on three sets of evaluation tasks in the ALFRED environment: (1) $EVAL_{INSTRUCT}$ to measure the ability of pre-trained agents to execute semantically meaningful instructions at varied levels of abstraction, (2) $EVAL_{LENGTH}$ to measure the ability of agents to chain behaviors across multiple trajectories to solve long tasks, and (3) $EVAL_{SCENE}$ to evaluate generalization performance when finetuning to unseen household floor plans. We did not use the official ALFRED benchmark test sets to construct $EVAL_{SCENE}$ since we require a task demonstration to compute how many subtasks the agent solved; these demonstrations are not given for the test set tasks. However, the tasks we evaluate on generally are designed to be representative of the tasks in the ALFRED test set: they test the agent on unseen instruction-scene combinations and consist of varied-length, compositional tasks. Like the ALFRED test set, our evaluation consists of long-horizon tasks that require sequential execution of multiple subtasks.

Collecting evaluation task data. The ALFRED dataset provides high-level language annotations for each of the trajectories in the dataset. We could use these annotations as unseen task-instructions to evaluate our agents. However, we found that the different skills are not equally distributed across trajectories of different skill lengths, e.g., most 2-skill trajectories perform pick-and-place tasks while tasks involving heating skills only appear in length 7+ trajectories. To allow evaluation with a less biased skill distribution,

we create the $EVAL_{INSTRUCT}$ task set by randomly choosing a trajectory from the ALFRED dataset *and then randomly sampling a subsequence of skills of a certain length* from this trajectory. To obtain a high-level language instruction that summarizes this new subsequence, we crowd-source labels from human annotators. For labeling, each annotator is presented with a remotely hosted Jupyter notebook interface (see Figure A.4). Whenever we by chance sample a full ALFRED trajectory for annotation, we directly used the existing high-level annotation from the ALFRED dataset. We annotate 80 trajectories with human annotators and combine them with 20 randomly sampled single-skill trajectories, resulting in a total of 100 evaluation tasks (see Figure A.5 for example instructions). This results in 20 tasks of length 1 skills, 20 tasks of length 2 skills, 20 tasks of length 3 skills, 20 tasks of length 4 skills, and 20 tasks of lengths 5+ (5-7) skills.

For $EVAL_{LENGTH}$, we randomly sampled 20 full trajectories from the ALFRED dataset that had sequences of 7 or 8 skills (10 of length 7, 10 of length 8) and removed these trajectories from the training dataset before performing LLM-based skill aggregation. This ensures AM and SPRINT must perform skill chaining to solve these tasks by ensuring that there were valid sequences of skills to chain together to be able to solve these removed tasks. For example, assume a (shortened for clarity) sampled skill sequence is "pick up apple," then "put apple in microwave", then "slice the apple." Then, either Actionable Models or SPRINT can chain together the sub-trajectory associated with "pick up apple" then "put apple in microwave" with the "slice the apple" sub-trajectory to solve this task. These trajectories all had annotations from ALFRED annotators, so we used those annotations directly (see Figure A.7 for example instructions).

Finally, for $EVAL_{SCENE}$, we collected a set of 10 full-length trajectories from the ALFRED "valid-unseen" dataset consisting of validation tasks in unseen floor plans. We collected 2 of each length from 1 through 5 for a total of 10 tasks by sampling random full-length trajectories from this dataset, with the exception of length 1 tasks (we just sample random skills to create length 1 tasks). As these are full trajectories,

they already have human annotations from ALFRED, which we directly use as the task description (see Figure A.6 for example instructions).

We list additional details about the tasks in each evaluation set in Table A.1.

Finally, we display 5 randomly sampled tasks, along with their human annotations, from each of our task sets in Figures A.5, A.6, and A.7.

Online finetuning environment setup. During online-finetuning we initialize the agent in the same house floor plan as the trajectory the task was extracted from to ensure executability. During finetuning, we give each episode a time horizon of 2x the number of environment actions needed by the expert planner to solve the task. We give sparse sub-task rewards for each skill solved by the agent during the episode. Therefore for length 1 tasks, the agent can only be rewarded once before the episode ends, while for length 5 tasks, the episode terminates on the fifth reward signal.

A.3.2 Real Robot

Here we detail the dataset and evaluation tasks used for the real world tabletop environment experiments.

Dataset Details. Part of our data comes from data collected from prior work on the same arm setup [76], in addition to additional trajectories collected for this project.

In total, we collected 329 long-horizon trajectories, resulting in ∼6k individual "primitive" skills consisting of pick and place tasks such as *"pick up the black bowl"* or *"put the apple in the sink."* These trajectories involve unique scene arrangements of different toy objects such as an apple, orange, black bowl, white plate, oven, sink, dish rack, etc. The total dataset size is 455,473 individual state-action pairs.

Evaluation Tasks. We formulate 3 unseen evaluation tasks requiring the completion of 2, 4, and 8 subtasks. These tasks are set in an environment configuration that has not been seen before in the training

data, i.e., the object combination is not present in the training data. For each of the tasks, we collect 25 demonstrations to finetune pre-trained policies for evaluation.

The three tasks are defined below:

1. Bake bread in the oven (length 2): The robot must (1) pick up the bread, (2) place it in the oven.
2. Serve heated milk in the bowl (length 4): The robot must (1) pick up the milk carton, (2) place it in the black bowl bowl, (3) pick up the bowl with the milk in it, (4) place the bowl in the oven.
3. Serve milk in the bowl and butter and baked bread in the plate (length 8): The robot must: (1) pick up the milk carton, (2) put it in the black bowl, (3) pick up the butter stick, (4) put it in the plate, (5) pick up the bread, (6) bake the bread in the oven, (7) pick up the bread from the oven, (8) place the bread in the plate.

A.4 Extended Experiments, Results, and Analysis

Table A.2: $EVAL_{INSTRUCT}$ and $EVAL_{LENGTH}$ eval dataset per-length and overall skill completion rates. See Section 3.4 for experiment setup.

		AM	ET	L-BC	SayCan	SPRINT
$EVAL_{INSTRUCT}$	Number of Completed Subtasks Overall	0.82 ± 0.07	1.15 ± 0.14	0.39 ± 0.02	1.00 ± 0.12	**1.94 ± 0.04**
	Length 1 Progress	0.47 ± 0.06	0.75 ± 0.07	0.89 ± 0.07	**0.94 ± 0.02**	0.89 ± 0.04
	Length 2 Progress	0.75 ± 0.10	1.20 ± 0.18	0.66 ± 0.05	0.69 ± 0.22	**1.52 ± 0.10**
	Length 3 Progress	0.96 ± 0.28	1.61 ± 0.23	0.27 ± 0.08	0.61 ± 0.09	**2.21 ± 0.03**
	Length 4 Progress	0.56 ± 0.17	1.45 ± 0.25	0.05 ± 0.05	0.60 ± 0.18	**2.36 ± 0.16**
	Length 5 Progress	1.59 ± 0.53	0.76 ± 0.14	0.07 ± 0.05	0.57 ± 0.04	**3.04 ± 0.24**
	Length 6 Progress	1.20 ± 0.40	0.86 ± 0.94	0.05 ± 0.08	0.24 ± 0.08	**2.87 ± 0.20**
	Length 7 Progress	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.40 ± 0.49**	0.00 ± 0.00
$EVAL_{LENGTH}$	Number of Completed Subtasks Overall	1.71 ± 0.43	1.76 ± 0.14	0.07 ± 0.00	0.66 ± 0.08	**4.40 ± 0.39**
	Length 7 Progress	0.80 ± 0.16	0.78 ± 0.45	0.06 ± 0.05	0.50 ± 0.06	**3.38 ± 0.43**
	Length 8 Progress	2.62 ± 0.71	2.74 ± 0.55	0.26 ± 0.18	1.50 ± 0.24	**5.25 ± 0.64**

Here, we present additional results complementary to the experiments in the main paper in Section 3.4. We present and analyze LLM annotation examples in Section A.4.1.

A.4.1 LLM Summary Examples

We randomly sample 3 LLAMA-13b task summaries produced while performing skill aggregation in ALFRED (explained in Section 3.3.2) using the prompt in Figure A.1 and display them in Figure A.8 along with summaries from OPT-350m and OPT-1.3B [408], 350M and 1.3B parameter open-source models for comparison. After analyzing many more examples, we see that LLaMA-13b generally provides fitting high-level summaries for most sequences by skipping over implied sub-tasks (although it sometimes also skips over important sub-tasks, likely due to the prompt). The other smaller models, which are also pre-trained on smaller corpora, tend to produce worse summaries and make up details more often.

A.4.2 Qualitative Comparison Results

Here we display and analyze some qualitative task execution examples from ALFRED and our real robot environment.

A.4.2.1 ALFRED

Zero-shot evaluation. We compare SPRINT, AM, and L-BC zero-shot evaluation results on long $EVAL_{LENGTH}$ tasks in Figure A.11. In general, SPRINT is able to make substantially more progress on $EVAL_{LENGTH}$ tasks as it leverages the large language model to generate longer-horizon, semantically meaningful pre-training tasks and performs cross-trajectory chaining to learn to chain its existing dataset tasks. In the visualized examples, SPRINT is able to understand and successfully execute many of the sub-tasks implied but not directly stated by the natural language task instruction. L-BC makes very little progress on these tasks, not even understanding what the first sub-task to complete should be as the task annotation is out of distribution from what it saw while training. Finally, AM is able to make some progress on some of these tasks due to its long-horizon goal pre-training objective. However, this is less effective than our language-conditioned pre-training in such zero-shot evaluations.

We show some example plans generated by SayCan, two that did not complete the task and one that did, in Figure A.9. While SayCan can generate correct plans for certain tasks, the plans generated are subject to failures by the LLM to pick the correct skill.

Finetuning. We finetune SPRINT, AM, and L-BC on $EVAL_{SCENE}$ tasks, in household floorplans that were never seen while training, and visualize qualitative policy rollout examples *after finetuning* in Figure A.12. In general, SPRINT is able to finetune to longer-horizon tasks while AM and L-BC both struggle with making progress on longer-horizon tasks despite receiving rewards for every completed sub-task. SPRINT's ability to complete more sub-tasks on many of the longer-horizon tasks is demonstrated in Figure A.12a, while a case in which both SPRINT and AM make partial progress throughout finetuning is demonstrated in Figure A.12b. We believe that AM has more trouble finetuning on these tasks than SPRINT because the task specification for AM (goal images) is out of distribution; pre-training on *language* tasks with

SPRINT allows agents to more easily learn longer-horizon behaviors as the task specifications may still be in-distribution of the pre-training tasks that LLM skill-aggregation and skill chaining produce.

We do not fine-tune SayCan as it is not a pre-training/fine-tuning method. This makes it susceptible to both planning and policy execution failures in the unseen environments in $EVAL_{SCENE}$. We demonstrate policy execution failures in Figure A.10.

A.4.2.2 Real Robot

We visualize evaluation rollouts after finetuning on our most difficult, length 8 task, *"Serve milk in the bowl and butter and baked bread in the plate,"* in Figure A.13. We display an example comparison between SPRINT and L-BC composite, the best-performing L-BC baseline in which the fine-tuned SPRINT model successfully follows and accomplishes the skills in the demonstrated long-horizon sequences. The L-BC composite agent finishes the first four skills before encountering confusion about the subsequent skill: pick up the long bread. This comparison reveals that the L-BC composite model exhibits proficiency in completing some skills but overall does struggle with long-horizon tasks. Empirically in our evaluations, we saw that this baseline exhibits greater variance than SPRINT among its evaluation runs, sometimes only executing 2 skills and other times finishing 8.

Instructions: give a high-level description for the following steps describing common household tasks.

Task Steps: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.
Summary: Put the box with keys on the sofa.

Task Steps: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.
Summary: Put a cooled slice of lettuce on the counter.

Task Steps: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.
Summary: Put a book on the couch.

Task Steps: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.
Summary: Put the cleaned fork in a drawer.

Task Steps: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.
Summary: Put the box of tissues on the barred rack.

Task Steps: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.
Summary: Put a heated glass on the wooden rack.

Task Steps: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.
Summary: Look at the box under the lamp light.

Task Steps: 1: [SKILL 1]. 2: [SKILL 2]. 3: [SKILL 3]. ... N: [SKILL N].
Summary:

Figure A.1: The full prompt that we use for summarization. Following the suggestions of Ahn et al. [7] for prompt design, we explicitly number each step. The LLM completion task begins after "Summary:". For brevity, we omit the new line characters between all numbered steps.

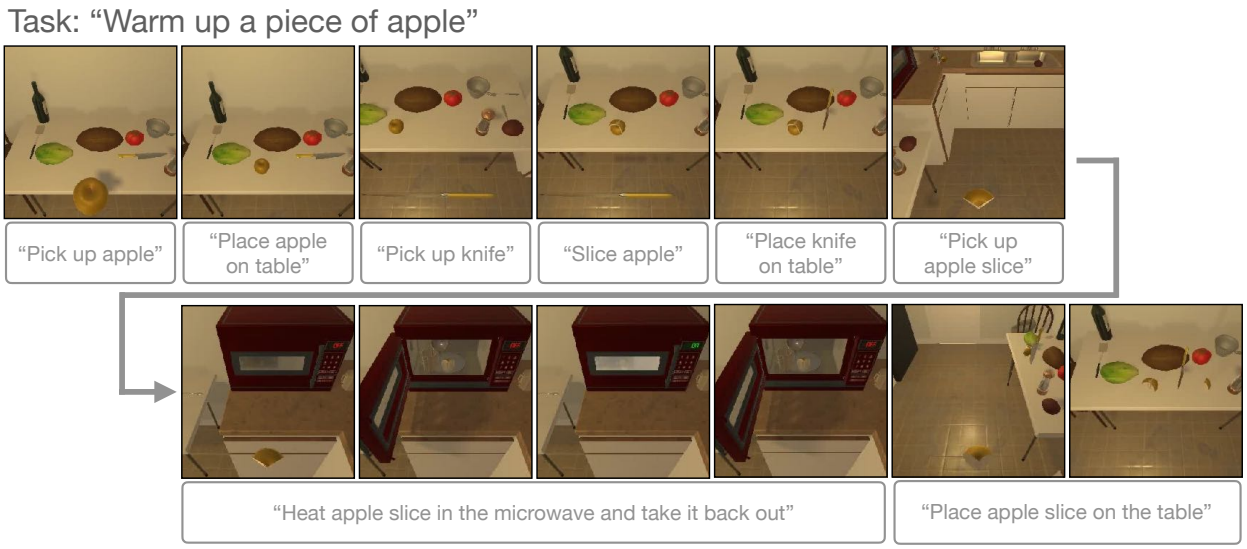
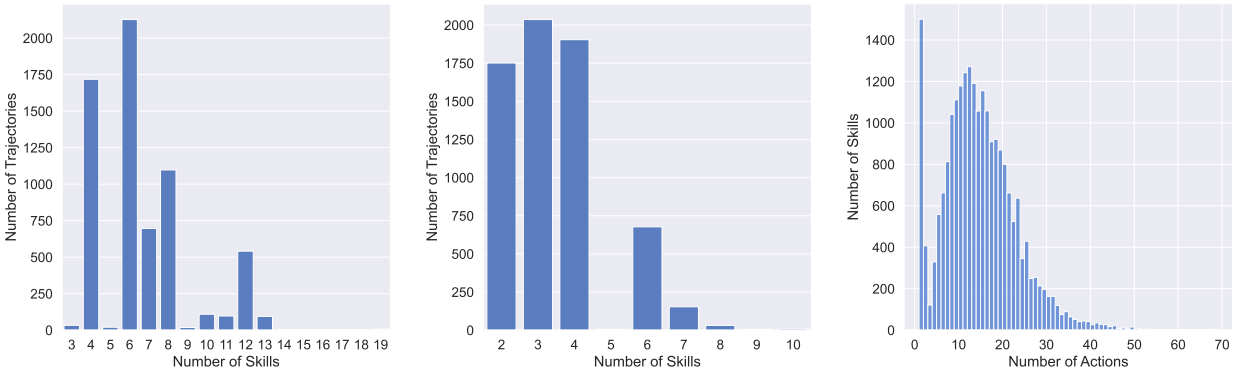


Figure A.2: Example successful task execution of our pre-trained SPRINT agent for the challenging “*Warm up a piece of apple*” task. Successful execution requires solving 8 subtasks in sequence and a total of 50 steps. This sequence of subtasks was never observed in the training data. SPRINT uses cross-trajectory stitching and LLM aggregation to learn unseen tasks.



(a) Skills per trajectory in the original ALFRED dataset.

(b) Skills per trajectory in the merged dataset.

(c) Actions per skill in the merged dataset.

Figure A.3: **Left**: distribution of the number of skills in each trajectory in the original ALFRED dataset. **Middle**: distribution of skills per trajectory in the “merged” dataset with merged navigation skills. **Right**: distribution of number of actions per skill in the “merged” dataset.

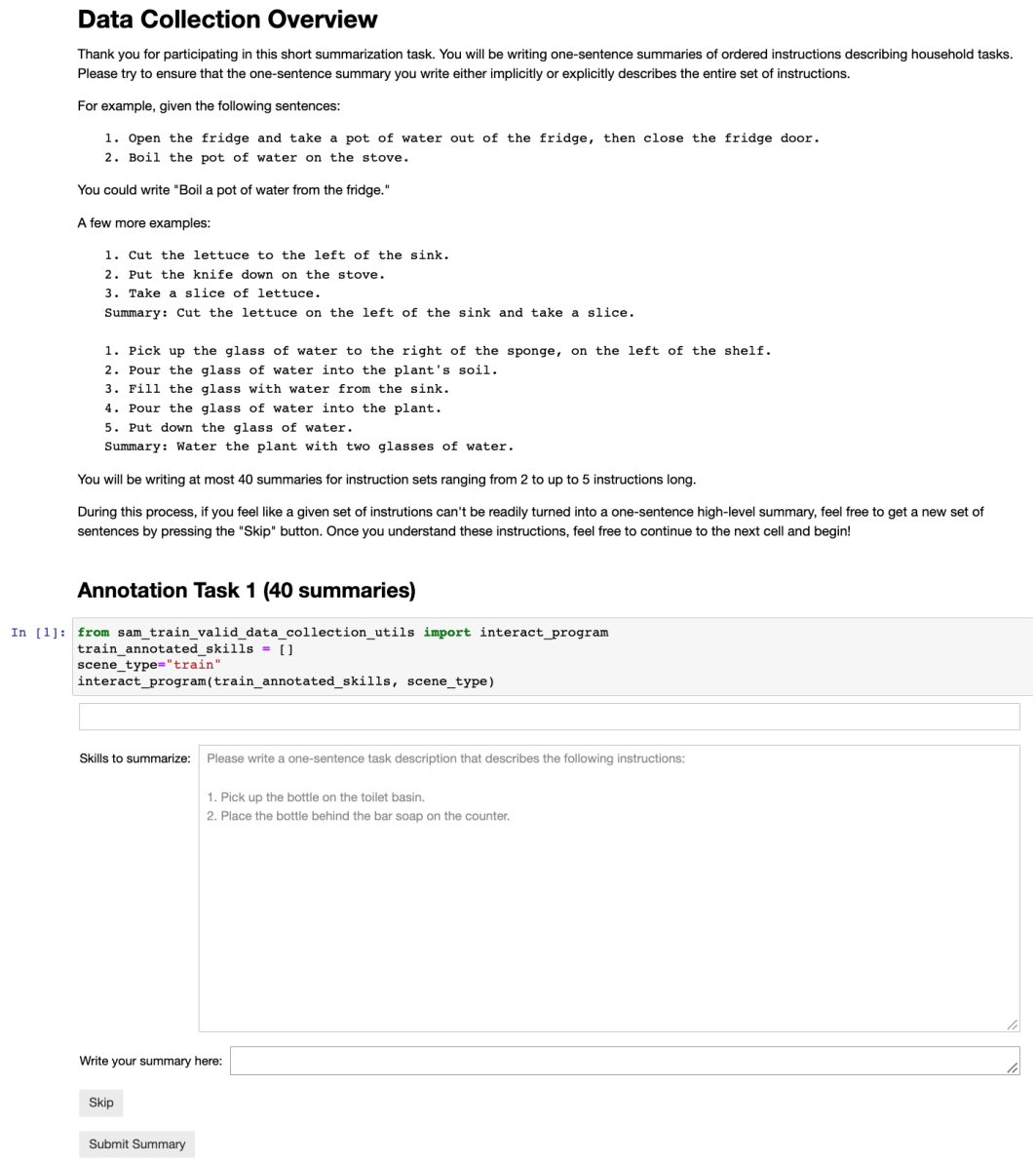


Figure A.4: Data collection jupyter notebook page. Note that there is a “Skip” button so that human annotators can skip an instruction sequence if they do not feel it is semantically meaningful or easy to summarize.

Skills to Summarize: 1: Grab the knife on the counter. 2: Place the knife in the sink then turn the faucet on so water fills the sink. Turn the faucet off and pick up the knife again. 3: Place the knife on the table to the left of the wooden bowl.
Annotator Summary: Wash the knife from the counter, put in on the table.

Skills to Summarize: 1: Pick up the blue book closest to your and the phone from the bed. 2: Turn on the lamp to take a look at the book in the light.
Annotator Summary: Examine the book by the light of a lamp.

Skills to Summarize: 1: Pick up yellow candle on counter. 2: Open cabinet, put candle in cabinet, close cabinet 3: Pick up yellow candle from toilet.
Annotator Summary: Move the candle from the sink to the cabinet under the sink, close it and and then pick the candle from the top of the toilet in front of you.

Skills to Summarize: 1: Pick the pot on the left side up from the stove. 2: Set the bowl and knife on the table next to the tomato.
Annotator Summary: Put the bowl with the knife in it next to the tomato.

Skills to Summarize: 1: Pick up the pen that's in front of you that's under the mug. 2: Put the pencil in the mug that was above it. 3: Pick up the mug with the pencil in it.
Annotator Summary: Put the pen into the mug and pick up the mug.

Figure A.5: Randomly sampled, human language instruction annotations from the $EVAL_{INSTRUCT}$ task set.

Skills to Summarize: 1: Pick up the lettuce on the counter. 2: Chill the lettuce in the fridge. 3: Put the chilled lettuce on the counter, in front of the bread.
Annotator Summary: Put chilled lettuce on the counter.

Skills to Summarize: 1: Pick up an egg from off of the kitchen counter. 2: Open the fridge, put the egg in to chill for a few seconds and then take it back out. 3: Place the cold egg in the sink.
Annotator Summary: Chill an egg and put it in the sink.

Skills to Summarize: 1: Pick up the butter knife off of the right side of the kitchen island. 2: Put the knife handle down in the frying pan that is on the front left burner of the stove. 3: Pick up the frying pan with the knife in it off of the stove. 4: Put the frying pan with the knife in it into the sink basin to the right of the potato.
Annotator Summary: Put a frying pan with a knife in it into the sink.

Skills to Summarize: 1: Take the pencil from the desk. 2: Put the pencil on the desk.
Annotator Summary: Take the pencil from the desk, put it on the other side of the desk.

Skills to Summarize: 1: Pick up the left pillow on the chair. 2: Put the pillow on the sofa right of the newspaper. 3: Pick up the pillow on the chair. 4: Put the pillow on the sofa left of the newspaper.
Annotator Summary: Place two pillows on a sofa.

Figure A.6: Randomly sampled, human language instruction annotations from the $EVAL_{SCENE}$ task set.

Skills to Summarize: 1: Pick up the knife in front of the lettuce. 2: Slice the apple in the sink with the knife. 3: Place the knife into the sink. 4: Pick up the sliced apple from the sink. 5: Place the apple slice into the pot on the stove. 6: Pick up the pot from the stove. 7: Pick ump the pot from the stove.
Annotator Summary: Slice an apple for the pot on the stove and put the pot on the counter to the right of the door.

Skills to Summarize: 1: Take the apple from the counter in front of you. 2: Place the apple in the sink in front of you. 3: Take the knife by the sink in front of you. 4: Cut the apple in the sink in front of you. 5: Place the knife in the sink in front of you. 6: Take an apple slice from the sink in front of you. 7: Heat the apple in the microwave, take it out and close the microwave. 8: Place the apple slice in the sink in front of you.
Annotator Summary: Place a warm apple slice in the sink.

Skills to Summarize: 1: Pick up the loaf of bread. 2: Put the bread on the counter above the spatula. 3: Pick up the knife that's above and to the right of the loaf of bread. 4: Cut the top half of the loaf of bread into slices. 5: Put the knife on the edge of the counter in front of you horizontally. 6: Pick up a slice of bread from the middle of the loaf. 7: Cook the bread in the microwave then take it out and close the microwave door. 8: Throw the cooked slice of bread away.
Annotator Summary: Put a microwaved slice of bread in the oven.

Skills to Summarize: 1: Pick the knife up from off of the table. 2: Open the microwave, slice the potato, and close the microwave. 3: Open the microwave, place the knife inside of it, and close the microwave. 4: Open the microwave, pick up the potato slice inside, close the microwave. 5: Place the potato slice in the pan on the stove. 6: Pick up the pan from the stove. 7: Open the refrigerator, place the pan inside, and close the refrigerator.
Annotator Summary: Move the pan from the stove top to inside the black refrigerator.

Skills to Summarize: 1: Pick up the red tomato on the counter to the right of the stove. 2: Put the tomato onto the island below the butter knife. 3: Pick up the butter knife off of the kitchen island. 4: Slice up the tomato on the kitchen island. 5: Place the butter knife onto the island to the right of the sliced tomato. 6: Pick up a tomato slice off of the kitchen island. 7: Open the fridge and put the tomato slice on the bottom shelf, then close the door, after a couple seconds open the fridge and remove the tomato slice then close the door. 8: Open the microwave door and place the tomato slice inside the microwave in front of the egg.
Annotator Summary: Put a chilled tomato slice into the microwave.

Figure A.7: Randomly sampled, human language instruction annotations from the $EVAL_{LENGTH}$ task set.

Skills to summarize: 1. Pick the clock up from the desk. 2. Turn the lamp on.
OPT-350m Summary: Look at the clock.
OPT-1.3b Summary: Look at the clock from the far side of the bed under the lamp light.
LLaMA-13b Summary: Look at the clock from the desk under the lamp light.

Skills to summarize: 1. Take the plate from the cabinet to the left of the stove. 2. Warm the plate in the microwave, above the stove. 3. Put the plate in the cabinet.
OPT-350m Summary: Place the plate in the microwave.
OPT-1.3b Summary: Put the plate on the left side of the table.
LLaMA-13b Summary: Put a warm plate in the cabinet above the stove.

Skills to summarize: 1. Open the microwave, take out the potato and shut the door. 2. Put the potato in the sink, turn the water on and off again. Take the potato out of the sink. 3. Place the potato on the counter, to the right of the sink.
OPT-350m Summary: Put the potato on the counter, to the left of the sink.
OPT-1.3b Summary: Open the microwave, take out the potato, and close the door.
LLaMA-13b Summary: Put a potato on the counter, next to the sink.

Figure A.8: Comparison of OPT-350m, OPT-1.3b [408], and LLaMA-13b summaries on 3 randomly sampled sequences from our dataset. In general, as the model size increases, the summary becomes better. LLaMA-13b summaries as a whole tend to be more descriptive and accurate. The smaller models tend to regurgitate information from the original skills incorrectly, such as OPT-350m summarizing the third example as putting a potato "to the left of the sink" when the original skill stated "to the right of the sink."

Task: Cut a potato and put a slice in the sink. (Fail)

Ground Truth	SayCan Generated Plan
1. Pick up the knife from the sink.	1. Pick up the knife from the sink.
2. Cut the potato on the second shelf in the fridge.	2. Put the knife back in the sink.
3. Put the knife back in the sink.	
4. Take a potato slice from the sink.	
5. Put the slice in the sink.	

Task: Move all computers onto the white dresser. (Fail)

Ground Truth	SayCan Generated Plan
1. Place the computer on the white dresser.	1. Place the computer on the white dresser.
2. Close the computer and pick up the computer from the green bed.	2. Place the computer on the white dresser.
3. Place the computer on the white dresser.	

Task: Move the phone from the dresser to the bed. (Success)

Ground Truth	SayCan Generated Plan
1. Take the blue cell phone off of the dresser.	1. Take the blue cell phone off of the dresser.
2. Put the blue cell phone on the bed.	2. Put the blue cell phone on the bed.

Figure A.9: Example plans from SayCan [7] evaluated on $EVAL_{INSTRUCT}$. For longer tasks SayCan has a higher probability of generating any single incorrect step in a plan, leading to planning failures that will prevent the language-conditioned policy from completing the task.

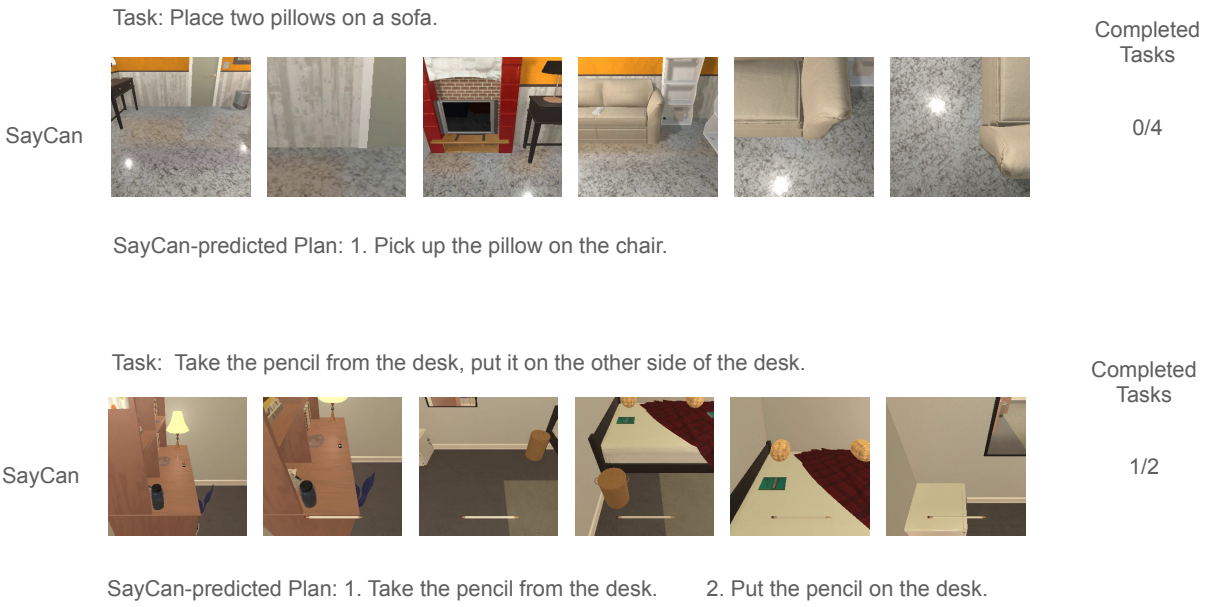
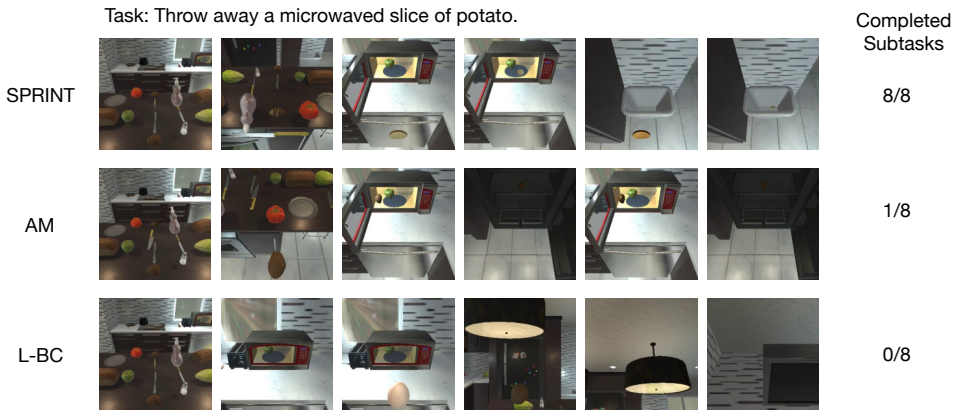
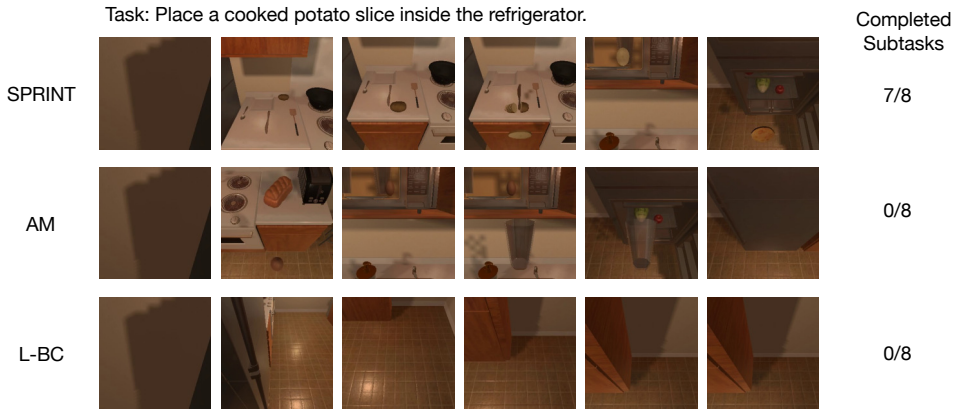


Figure A.10: Rollouts of SayCan on $EVAL_{SCENE}$. In these examples, SayCan predicts correct plan steps until the policy suffers from execution errors as it is not fine-tuned for the unseen environments.



(a) SPRINT successfully solves this task, while AM fails to slice the potato and repetitively iterates between putting the potato in the fridge and microwave. L-BC fails even to pick up the potato, as the task annotation does not directly describe picking up a potato.

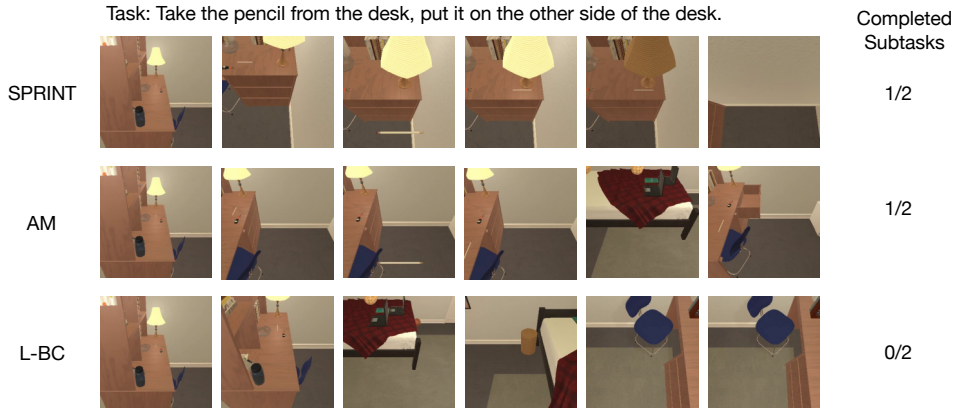


(b) SPRINT nearly solves this task, while AM picks up an egg instead of a potato. L-BC picks up random objects not related to the annotation.

(c) SPRINT completes the entire task. AM picks up the tomato but fails to put it down onto the counter and slice it. L-BC aimlessly wanders and picks up random objects.

Figure A.11: Visualizations of zero-shot policy rollouts on three tasks in the $EVAL_{LENGTH}$ task set.

(a) SPRINT picks up and places one of the pillows on the sofa, and picks up the second but does not manage to place the second on the sofa, thus completing 3/4 subtasks. AM and L-BC both learn to pick up a pillow but never learned to place it in the correct spot.



(b) SPRINT and AM both learn to pick up a pencil from the desk, although neither manage to put the pencil down in the correct place "on the other side of the desk." Meanwhile, L-BC never picks up the pencil.

Figure A.12: Visualizations of policy rollouts on two tasks in the $EVAL_{SCENE}$ task set, after finetuning each method. These floor plans were originally unseen to all agents until finetuning.

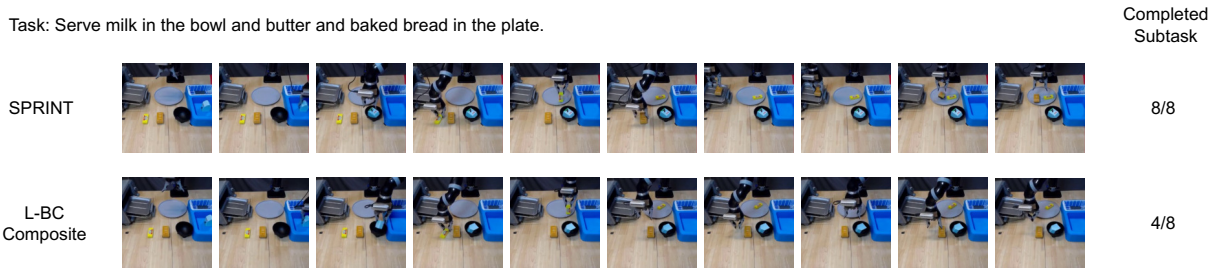


Figure A.13: SPRINT picks up the correct objects successfully and places in the right place accurately, with the same order shown in the demonstration. L-BC composite model does the right thing on the milk diary and butter diary but is not able to finish any skills with the long bread.

Appendix B

EXTRACT

B.1 Full Algorithm

Algorithm 3 EXTRACT Algorithm, Section 4.4.

Require: Dataset $\mathcal{D}$, VLM, Target MDP $\mathcal{M}$, Optional target task fine-tuning dataset $\mathcal{D}_{\mathcal{M}}$

1: $\mathcal{D}_d, CM \leftarrow$ OfflineSkillExtraction($\mathcal{D}$, VLM) ▷ Get discrete skill labels and clustering model, Algorithm 4
2: Init $q(z \mid \bar{a}, d), p_a(\bar{a} \mid z, d), p_d(d \mid s), p_z(z \mid s, d)$ ▷ Skill argument encoder, skill decoder, discrete skill prior, continuous argument prior
3: $q, p_a, p_d, p_z \leftarrow$ OfflineSkillLearning($\mathcal{D}_d$, q, p_a, p_d, p_z) ▷ Learn skills offline, Algorithm 5
4: **if** $\mathcal{D}_{\mathcal{M}}$ exists **then**
5: $\quad \mathcal{D}_{\mathcal{M},d} \leftarrow$ Assign skills to $\mathcal{D}_{\mathcal{M}}$ with existing clustering model CM
6: $\quad q, p_a, p_d, p_z \leftarrow$ OfflineSkillLearning($\mathcal{D}_{\mathcal{M},d}$, q, p_a, p_d, p_z) ▷ Optionally fine-tune on target task $\mathcal{M}$
7: SkillBasedOnlineRL($\mathcal{M}, p_a, p_d, p_z$) ▷ RL on target task $\mathcal{M}$, Algorithm 6

We present the full EXTRACT pseudocode in Algorithm 3. Algorithm 4 details offline skill extraction using a VLM, Algorithm 5 details the offline skill learning procedure, and Algorithm 6 details how to perform online skill-based RL on downstream tasks using Soft Actor-Critic (SAC). Note that any entropy-regularized algorithm can be used here with similar modifications, not just SAC. Differences from SAC during online RL are highlighted in red. For further implementation details and hyperparameters of EXTRACT, see Appendix B.2.1.

Algorithm 4 Offline Skill Extraction, Section 4.4.1.

procedure OFFLINESKILLEXTRACTION($\mathcal{D}$, VLM)
  EMBEDS $\leftarrow$ [] ▷ Init VLM embedding differences
  for trajectory $\tau = [(s_1, a_1), ..., (s_T, a_T)]$ in $\mathcal{D}$ **do**
    for (s_i, a_i) in τ **do**
      $e_i = \text{VLM}(s_i) - \text{VLM}(s_1)$ ▷ Embedding differences, Equation (4.1)
      EMBEDS.APPEND(e_i)
  $CM \leftarrow$ Init (K-Means) clustering model
  LABELS $\leftarrow CM$(EMBEDS) ▷ Run unsupervised clustering to get cluster labels
  $\mathcal{D}_d \leftarrow \{\}$ ▷ Init skill labeled dataset
  for trajectory $\tau = [(s_1, a_1), ..., (s_T, a_T)]$ in $\mathcal{D}$ **do**
    $d_1, ..., d_T \leftarrow$ Get labels from LABELS
    $d_1, ..., d_T \leftarrow$ MEDIANFILTER($d_1, ..., d_T$) ▷ Smooth out labels, see Appendix B.2.1
    $\mathcal{D}_d \leftarrow \mathcal{D}_d \cup [(s_1, a_1, d_1), ..., (s_T, a_T, d_T)]$
  return $\mathcal{D}_d, CM$

Algorithm 5 Offline Skill Learning, Section 4.4.2.

procedure OFFLINESKILLLEARNING($\mathcal{D}$, q, p_a, p_d, p_z)
  while not converged **do**
    Sample τ_d from $\mathcal{D}_d$
    Train q, p_a, p_d, p_z with Equation (4.2)
  return q, p_a, p_d, p_z

B.2 Experiment and Implementation Details

In this section, we list implementation details for EXTRACT (Appendix B.2.1), the specific environment setups (Appendix B.2.3), and details for how we implemented baselines (Appendix B.2.2).

B.2.1 EXTRACT Implementation Details

EXTRACT implementation details follow in the same order as each method subsection was presented in the main paper in Section 4.4.

B.2.1.1 Offline Skill Extraction

We first extract skills from a dataset $\mathcal{D}$ using a VLM by clustering VLM embedding differences of image observations in $\mathcal{D}$ (see pseudocode in Algorithm 4).

Algorithm 6 Skill-Based Online RL (with SAC [123]), Section 4.4.3. Red marks policy and critic loss differences against SAC.

procedure SkillBasedOnlineRL($\mathcal{M}, p_a(\bar{a} \mid z, d), p_d(d \mid s), p_z(z \mid s, d)$) ▷ Section 4.4.3
  Freeze $p_a(\bar{a} \mid z, d), p_d, p_z$ weights
  $\pi_d(d \mid s) \leftarrow p_d(d \mid s)$ ▷ Init π_d as discrete skill prior p_d
  $\pi_z(z \mid s, d) \leftarrow p_z(z \mid s, d)$ ▷ Init π_z as cont. argument prior p_z
  $B \leftarrow \{\}$ ▷ Init buffer B
  for each rollout **do**
    $l \leftarrow 0$
    $d_t \sim \pi_d(d \mid s_t)$ ▷ Sample discrete skill
    $z_t \sim \pi_z(z \mid s, d_t)$ ▷ Sample continuous argument for skill
    $a_1, ..., a_L, l_1 ... l_L \leftarrow \bar{a} \sim p_a(\bar{a} \mid z_t, d_t)$ ▷ Sample action sequence $a_1, ..., a_L$ and progress predictions $l_1, ..., l_l$ up to max sequence length L, see ??.
    for a in $a_1., ..., a_L$ or until $l \geq 1$ **do**
      Execute actions in $\mathcal{M}$, accumulating reward sum $\tilde{r}_t$
    $B \leftarrow B \cup \{s_t, z_t, \tilde{r}_t, s_{t'}\}$ ▷ Add sample to buffer
    $(s, z, \tilde{r}, s') \sim B$ ▷ Sample from B
    $\pi_d, \pi_z \leftarrow \max_{\pi_d, \pi_z} Q(s, z, d)$
      $-\alpha_z \text{KL}(\pi_z(z \mid s, d) \parallel p_z(\cdot \mid s, d))$
      $-\alpha_d \text{KL}(\pi_d(d \mid s) \parallel p_d(\cdot \mid s))$ ▷ Update policies, Equation (4.4)
    $Q \leftarrow \min_Q Q(s, z, d) = r(s, z, d) + \gamma Q(s', z', d')$
      $-\alpha_z D_{KL}(\pi(z \mid s, d) \parallel p_z(\cdot \mid s, d))$
      $-\alpha_d D_{KL}(\pi(d \mid s) \parallel p_d(\cdot \mid s))$ ▷ Update critic

Clustering. We use K-means for the clustering algorithm as it is performant, time-efficient, and can be easily utilized in a batched manner if all of the embeddings are too large to fit in memory at once.* When extracting skills from the offline dataset $\mathcal{D}$, we utilize K-means clustering on VLM embedding differences with $K = 8$ in Franka Kitchen and LIBERO, as we found $K = 8$ to produce the most visually pleasing clustering assignments in Franka Kitchen and we directly adapted the Franka Kitchen hyperparameters to LIBERO to avoid too much environment-specific tuning. In FurnitureBench, we found $K = 6$ to produce the most visually distinguishable clustering assignments.

Median Filtering. After performing K-means, we utilize a standard median filter, as is commonly performed in classical speaker diarization [16], to smooth out any possibly noisy assignments (see Figure 4.3). Specifically, we use the Scipy `scipy.signal.medfilt(kernel_size=7)` [361] filter for all environments. This corresponds to a median filter with window size 7 that slides over each trajectory's labels and assigns the median label within that window to all 7 elements. Empirically, we found that this increased the average length of skills as it reduced the occurrence of short, noisy assignments.

B.2.1.2 Offline Skill Learning

Here, we train a VAE consisting of skill argument encoder $q(z \mid \bar{a}, d)$, skill decoder $p_a(\bar{a} \mid z, d)$, discrete skill prior $p_d(d \mid s)$, and continuous skill argument prior $p_z(z \mid s, d)$ (see pseudocode in Algorithm 5).

Model architectures. We closely follow SPiRL's model architecture implementations [282] as we build upon SPiRL. The encoder $q(z \mid \bar{a}, d)$ and decoder $p_a(\bar{a} \mid z, d)$ are implemented with recurrent neural networks. The skill priors are both standard multi-layer perceptrons. The skill argument space z has 5 dimensions. In Kitchen and LIBERO, our β for the β-VAE KL regularization term in Equation (4.2) is 0.001.

*We did perform preliminary experiments early on with DBSCAN, which doesn't require presetting the number of clusters. However, DBSCAN requires an ϵ parameter which we found to greatly affect the skill clustering results on our datasets, with some values of ϵ resulting in very poorly clustered skills.

Skill progress predictor. During training, for GPU memory reasons, we sample skill trajectories with a maximum length as is common when training autoregressive models. In Franka Kitchen, this is heuristically set to 30 based on reconstruction losses and in LIBERO, this is set to 40. In FurnitureBench, this is set to 30. If a skill trajectory is longer than this maximum length, we simply sample a random contiguous sequence of the maximum length within the trajectory. To ensure that predicted action sequences stay in-distribution with what was seen during training, we also use these maximum lengths as maximum skill lengths during online RL; e.g., if a skill runs for 30 timesteps in Franka Kitchen without stopping, we simply resample the next skill (see Algorithm 6).

As discussed in Section 4.4.2, given the variable lengths of action sequences $\bar{a}$, the decoder $p_a(\bar{a} \mid z, d)$ is trained to generate a continuous skill progress prediction value l at each timestep. This value represents the proportion of the skill completed at the current time. During online policy rollouts, the execution of the skill is halted when l reaches 1. To learn this progress prediction value, we formulate it as follows: when creating labels for such a sequence, we assign a label to each time step, denoted as y_t, based on its position in the sequence. Specifically, y_t is set to $\frac{t}{N}$ for each time step t, where N represents the sequence length. To train the model for this function, we use the standard mean-squared error loss. This ensures that the model learns to predict the end of an action sequence while also ensuring that it receives dense, per-timestep supervision while training function.

Additional target task fine-tuning. Optionally, for very difficult tasks, some target-task demonstrations may be needed [355, 210, 201]. We perform additional target task fine-tuning in LIBERO [201] and FurnitureBench [135]. We use the learned clustering model that was trained to cluster the original dataset $\mathcal{D}$ to directly assign labels to the task-specific dataset $\mathcal{D}_{\mathcal{M}}$ without updating the clustering algorithm parameters (see Algorithm 3 Line 5). Then, we fine-tune the entire model, q, p_a, p_d, p_z, with the same objective in Equation (4.2) on the labeled target-task dataset $\mathcal{D}_{\mathcal{M},d}$.

B.2.1.3 Skill-Based Online RL

For online RL, we utilize the pre-trained skill decoder $p_a(\bar{a} \mid z, d)$, and the skill priors $p_d(d \mid s), p_z(z \mid s, d)$ for skill-based policy learning (see Algorithm 6).

Policy learning. Our policy skill-based policy $\pi(d, z \mid s)$ is parameterized as a product of a discrete skill selection policy $\pi_d(d \mid s)$ and a continuous argument selection policy $\pi_z(z \mid s, d)$ (see Equation (4.4)). To train with actor-critic RL, we sum over the policy losses in each discrete skill dimension weighted by the probability of that skill, similar to discrete SAC loss proposed by Christodoulou [61]:

$$\sum_d \pi_d(d \mid s)\Big(Q(s, z, d) - \alpha_z \mathrm{KL}(\pi_z(z \mid s, d) \parallel p_z(\cdot \mid s, d)) - \alpha_d \mathrm{KL}(\pi_d(d \mid s) \parallel p_d(\cdot \mid s))\Big). \tag{B.1}$$

Meanwhile, critic losses are computed with the skill d that the policy actually took. Our critic networks $Q(s, z, d)$ take the image s and argument z as input and have a d-headed output for each of the d skills.

We do not use automatic KL tuning (standard in SAC implementations [123]) as we found it to be unstable; instead, we manually set entropy coefficients α_d and α_z for the policy (Equation (4.4)) and critic losses. In Kitchen, $\alpha_d = 0.1$, $\alpha_z = 0.01$; in LIBERO $\alpha_d = 0.1$, $\alpha_z = 0.1$. These values are obtained by performing a search over $\alpha_d = \{0.1, 0.01\}$ and $\alpha_z = \{0.1, 0.01\}$.

In FurnitureBench, we set $\alpha_z = 0.5$ and $\alpha_d = 2.0$ to prevent the policy losses from diverging significantly as we use RLPD [23] with a high critic update ratio of 2 per environment step and a higher policy update ratio of 2 per environment step.

B.2.2 Baseline Implementation Details

Oracle. Our oracle baseline is RAPS [72]. We run RAPS to convergence and report final performance numbers because its expert-designed skills operate on a different control frequency; it takes hundreds of times more low-level actions per environment rollout. We only evaluated this method on Franka Kitchen

as the authors did not evaluate on our other environments, and we found the implementation and tuning of their hand-designed primitives to work well on other environments to be non-trivial and difficult to make work.

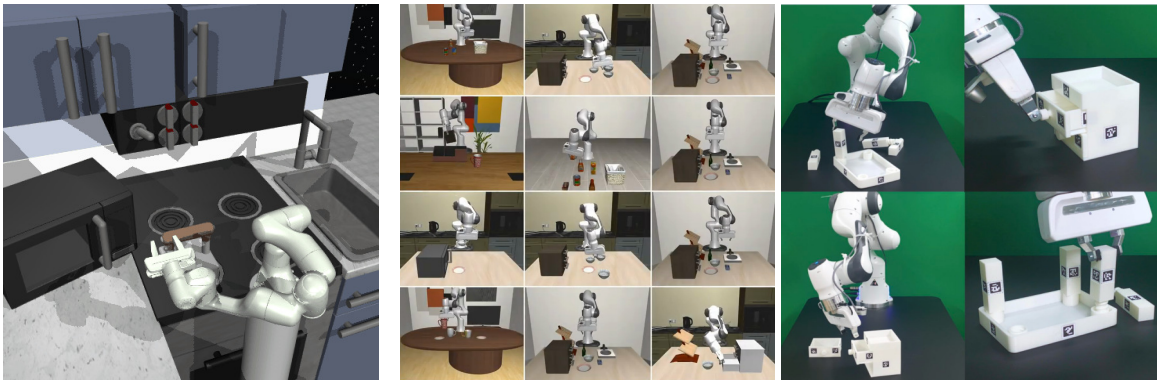
SPiRL. We adapt SPiRL, implemented on top of SAC [123], to our image-based settings and environments using their existing code to ensure the best performance. For each environment, we tuned SPiRL parameters (entropy coefficient, automatic entropy tuning, network architecture, etc.) first and then built our method upon the final SPiRL network architecture to ensure the fairest comparison. SPiRL uses the exact same datasets as ours but without skill labels. We also experimented with changing the length of SPiRL action sequences, and similar to what was reported in Pertsch, Lee, and Lim [282], we found that a fixed length of 10 worked best. We also found fixed prior coefficients KL divergence to perform better with SPiRL for our environments than automatic KL tuning.

EXTRACT-UVD. Universal Value Decomposer (UVD) segments trajectories into sub-trajectories using VLM features for image goal-conditioned behavior cloning [409]. It was originally made for goal-conditioned imitation learning; we combine it with EXTRACT and adapt it for our setting of online reward-based reinforcement learning by using it to segment subtrajectories with the same VLM as EXTRACT, then treating each subtrajectory as a separate skill trajectory to condition EXTRACT's model on (without discrete skill extraction process). Essentially, this model acts as EXTRACT but with skill trajectories determined by UVD's trajectory segmentation method instead of that of EXTRACT. This also makes the comparison against our method more fair as it receives temporally extended skills, just like SPiRL or EXTRACT.

BC. We implement behavior cloning with network architectures similar to ours and using the same datasets. Our BC baseline learns an image-conditioned policy $\pi(a \mid s)$ that directly imitates single-step environment actions. We fine-tune pre-trained BC models for online RL with SAC [123].

SAC. We implement Soft-Actor Critic [123] directly operating on low-level environment actions with an identical architecture to the BC baseline. It does not pre-train on any data.

B.2.3 Environment Implementation Details



(a) Franka Kitchen (b) LIBERO (c) FurnitureBench

Figure B.1: Our two image-based, continuous control robotic manipulation evaluation domains. **(a) Franka Kitchen:** The robot must learn to execute an unseen sequence of 4 sub-tasks in a row. **(b) LIBERO:** We evaluate 4 task suites of 10 tasks, each consisting of long-horizon, unseen tasks with new object, spatial, and goal transfer scenarios. **(c) FurnitureBench:** We evaluate online RL adaptation to unseen object and gripper placement randomizations.

Franka Kitchen. We use the Franka Kitchen environment from the D4RL benchmark [104] originally published by Gupta et al. [118] (see Figure B.1a). The pre-training dataset comes from the "mixed" dataset in D4RL consisting of 601 human teleoperation trajectories each performing 4 subtasks in sequence in the environment (e.g., open the microwave). Our evaluation task comes from Pertsch, Lee, and Lim [282], where the agent has to perform an *unseen* sequence of 4 subtasks. The original dataset contains ground truth environment states and actions; we create an image-action dataset by resetting to ground truth states in the dataset and rendering the corresponding images. For all methods, we perform pre-training and RL

with 64x64x3 RGB images and a framestack of 4. Sparse reward of 1 is given for each subtask, for a maximum return of 4. The agent outputs 7-dimensional joint velocity actions along with a 2-dimensional continuous gripper opening/closing action. Episodes have a maximum length of 280 timesteps.

LIBERO. LIBERO [201] is a continual learning benchmark built upon Robosuite [420] (see Figure B.1b). For skill extraction and policy learning, we use the `agentview_rgb` 3rd-person camera view images provided by the LIBERO datasets and environment. For pre-training, we use the LIBERO-90 pre-training dataset consisting of 4500 demonstrations collected from 90 different environment-task combinations each with 50 demonstrations. We condition all methods on 84x84x3 RGB images with a framestack of 2 along with language instructions provided by LIBERO. We condition methods on language by embedding instructions with a pre-trained, frozen sentence embedding model [298], `all-MiniLM-L6-v2`, to a single 384-dimensional embedding and then feeding it to the policy. For EXTRACT, we condition on language by conditioning all networks on language; q, p_z, p_a, p_d are all additionally conditioned on the language embedding and thus the skill-based policy is also conditioned on language. We also condition all networks in all baselines on this language embedding in addition to their original inputs.

When performing additional fine-tuning to LIBERO-{`10, Goal, Spatial, Object`}, for all methods (except SAC) we use the given task-specific datasets each containing 50 demonstrations per task before then performing online RL. In LIBERO-{`Goal, Spatial, Object`}, sparse reward is provided upon successfully completing the task, so the maximum return is 1.0. In LIBERO-10, tasks are longer-horizon and consist of two subtasks, so we provide rewards at the end of each subtask for a maximum return of 2.0. Episodes have a max length of 300 timesteps.

FurnitureBench. FurnitureBench [135] is a real-world furniture assembly benchmark, where the task is to assemble 3D printed furniture pieces with a single Franka Arm (see Figure B.1c). We closely reproduced

the environment setup presented in the original paper through manual camera calibration of RealSense D435 cameras. For skill extraction and policy learning, we use two cameras, a wrist-mounted camera and a front mounted camera facing the arm workspace. To cluster skills, we embed both the wrist camera and front camera images with R3M and concatenate the embedding before clustering with K-Means ($K = 6$). Following the baselines implemented in the paper, we encode all RGB images with the frozen R3M video encoder to a 2048 dimensional vector first for all components of skill/policy learning. Also following the paper, we don't use any framestacking. Additionally, despite the presence of AprilTags, we do not use AprilTag-based state estimation for any part of our experiments; we perform purely image-based continuous control.

The pre-training dataset consists of 500 demonstrations from the `one-leg` assembly task collected by Heo et al. [135]. See Figure B.2 for an image of the pieces used. The environment action space is absolute 3D position control plus a 6-dimensional rotation representation [414]. We run real-world online RL with RLPD [23], a sample-efficient actor-critic algorithm that uses a high critic update ratio, layer norms in the Q functions, a large number of Q functions, and samples from offline data and new online-collected data at a 50/50 ratio. We did not have to modify the policy/training objectives in Algorithm 6 from SAC for RLPD. To train EXTRACT's high-level policy with offline data in RLPD, we use the skill d in the offline trajectories as high-level skill selection actions for $\pi(d \mid s)$ and encode offline sampled trajectory actions with the pre-trained, frozen encoder $q(z \mid \bar{a}, d)$ to obtain continuous arguments z for $\pi(z \mid s, z)$. The offline data also comes with sub-task completion timestamps (picked up table top, placed it into the corner, picked up leg, inserted leg, screwed in) that we convert into $+1$ rewards for RLPD training for a maximum return of 5.

Figure B.2: 3D-printed FurnitureBench table used for our `one leg` assembly task.

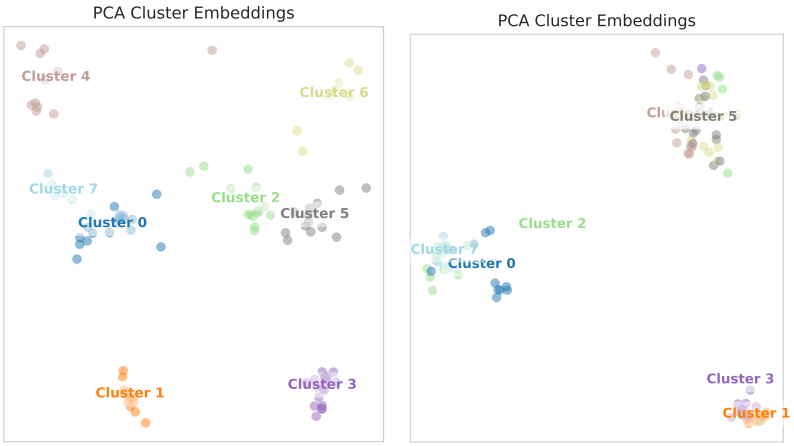
In our real-world RL setup, we run all methods for 100 trajectories with variations of 5cm for the initial object positions and end effector positions, along with ± 15 degrees of initial end effector rotation. Meanwhile, we use a dataset of 500 demonstrations from the "low randomness" dataset split for `one-leg` assembly from [135], which contains no intentional randomness for both the object and end effector poses. This is challenging as the robot arm, camera positioning, etc., and now initial object and end effector locations, are all different from those in the dataset as collected by the FurnitureBench authors and the policy must *transfer* its knowledge to this new setting to solve the task. We provide two rewards when training online RL: $+1$ for completing an assembly sub-task successfully and 0 for all other timesteps. The max return is also 5.

Episodes have a maximum length of 500 timesteps. Each trajectory takes approximately 50s of robot interaction time when run to completion and 30s to reset, resulting in $\sim$1.5 minutes of real-world time per trajectory.

B.3 Additional Experiments and Qualitative Visualizations

In this section, we perform additional experiments and ablation studies. In Appendix B.3.1, we visualize 2D PCA plots of clusters generated by EXTRACT in all environments. In Appendix B.3.2, we analyze statistics of the skill distributions generated by EXTRACT. In Appendix B.3.3 for more ablation studies comparing using CLIP [289] or proprioceptive states instead of R3M [252] for clustering feature extraction. Finally, in Appendix B.3.4 we analyze skills extracted through UVD [409] against ours.

B.3.1 Additional PCA Cluster Visualizations

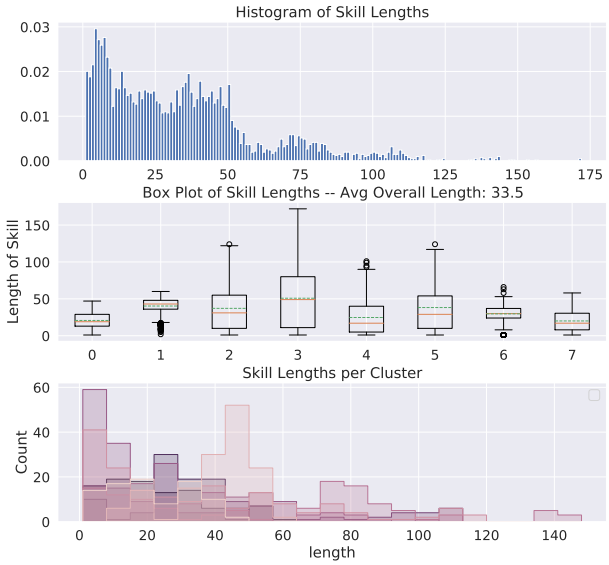


(a) Franka Kitchen. (b) LIBERO.

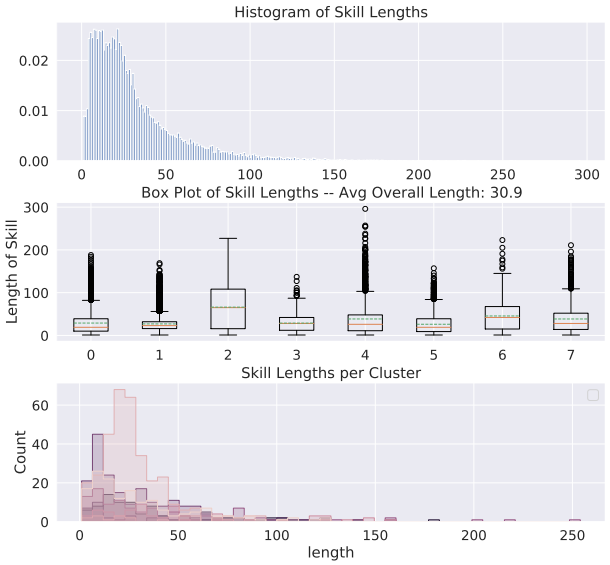
Figure B.3: 100 randomly sampled trajectories from environment pre-training datasets after being clustered into skills and visualized in 2D with PCA. Clusters are well-separated, even in just 2-dimensions with a linear transfromation.

Here we display PCA skill cluster visualizations in Figure B.3. Franka Kitchen clusterings are very distinguishable, even in 2 dimensions. (this is the same embedding plot as in Figure 4.4 in the main paper). LIBERO-90 clusters still demonstrate clear separation, but are not as separable after being projected down to 2 dimensions (from 2048 original dimensions). However, in Figure B.8 we clearly see distinguishable behaviors among different skills in LIBERO.

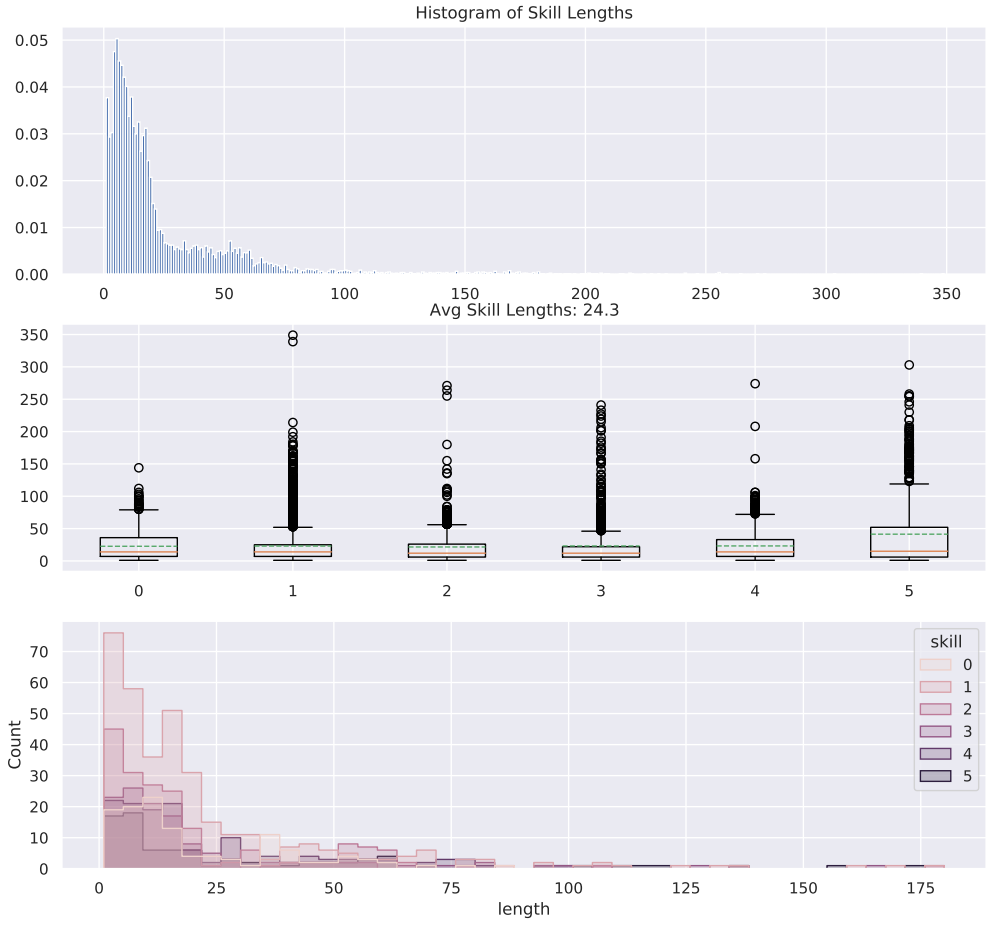
B.3.2 Visualizing Cluster Statistics



(a) Franka Kitchen, $K = 8$.



(b) LIBERO, $K = 8$.



(c) FurnitureBench, $K = 6$.

Figure B.4: Skill/clustering statistics in all environments. We use the R3M VLM [252] and $K = 8$ for K-means. In FurnitureBench, $K = 6$. The top plots are skill length histograms for all skill trajectories combined, middle plots correspond to box-and-whisker plots with skill ID on the x-axis and lengths on the y-axis, and the bottom plots represent distributions of skill lengths separated by color for each skill ID.

We visualize skill clustering statistics in all pre-training environments in Figure B.4. The plots demonstrate that average skill lengths are about 30 timesteps for all environments and that there is clear separation among the different skills just in terms of the distributions of skill lengths that they cover. For a qualitative look at the skills, see Appendix B.4.

B.3.3 Additional Ablation Studies

Here, we augment Section 4.5.4 with additional ablation studies. We compare against using **CLIP** [289] embeddings differences instead of R3M and against **Proprio**, i.e., robot joint and gripper state differences, on Franka Kitchen in Figure B.5.

We originally chose R3M as the base VLM because, in contrast with R3M, CLIP is trained on images, not videos, on a general dataset of image-language pairs instead of a dataset of humans performing real-world tasks from an egocentric viewpoint that more closely mimics robotics environments (Ego4D, [112]). However, we can see that EXTRACT with CLIP performs on par with EXTRACT with R3M. This demonstrates that EXTRACT is robust to the choice of VLM for clustering; it's likely that CLIP was pre-trained on sufficient data to extract useful embedding differences for clustering. However, proprioceptive state differences are not as effective as proprioception can be difficult to directly obtain *high-level*, semantically meaningful skills from.

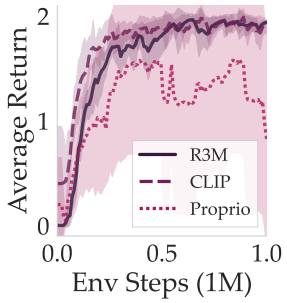


Figure B.5: EXTRACT with R3M vs with CLIP or proprioceptive states.

B.3.4 Visualizing UVD's Skill Extraction vs Ours

In Section 4.5.3 we found EXTRACT with UVD's skill extraction method to have unstable RL performance in Franka Kitchen and overall performed worse than EXTRACT with our skill extraction method.

To analyze why, we plot R3M skill embeddings of skill trajectories extracted by EXTRACT against those of UVD, projected to 2 dimensions, in Franka Kitchen in Figure B.6. We can see that UVD-generated trajectory embeddings are much harder to distinguish from each other than EXTRACT's, as evidenced by the distinct separation seen in the 4 corners of the plot compared to UVD.

This difference in trajectory separability, combined with EXTRACT's skill clustering approach that forms a discrete-continuous skill space for RL policies to learn new tasks with, helps explain the instability of RL with UVD in Franka Kitchen.

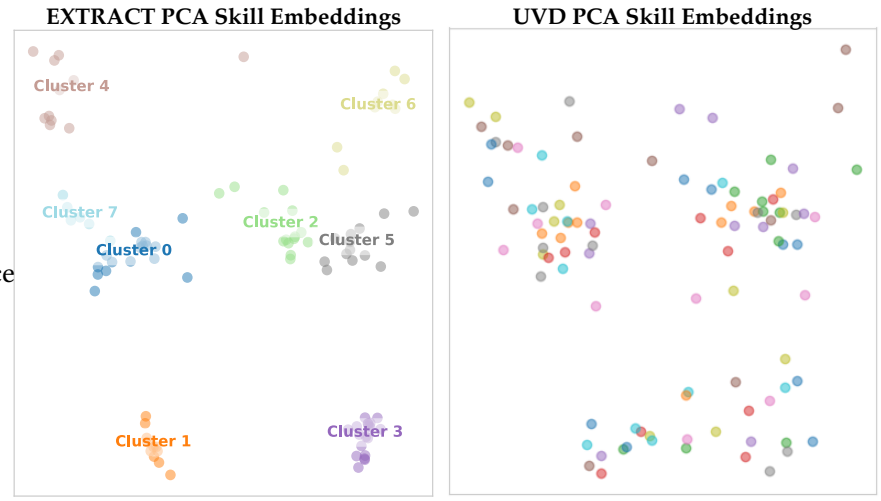


Figure B.6: 100 randomly sampled trajectories from Franka Kitchen after being projected to 2D with PCA. EXTRACT embeddings are identical to those in Figure B.3.

B.4 Visualizing skill trajectories

Here, we visualize skill trajectories in our environments. In Figure B.7, we visualize purely randomly sampled clusters (i.e., without any cherry-picking) in Franka Kitchen, where we see skills are generally semantically aligned. For example, skill 3 trajectories correspond to manipulating knobs, skill 5 trajectories reach for the microwave door, and skill 7 trajectories are reaching for the cabinet handle.

We visualize LIBERO skills in Figure B.8, where we can also see that skills are generally aligned.

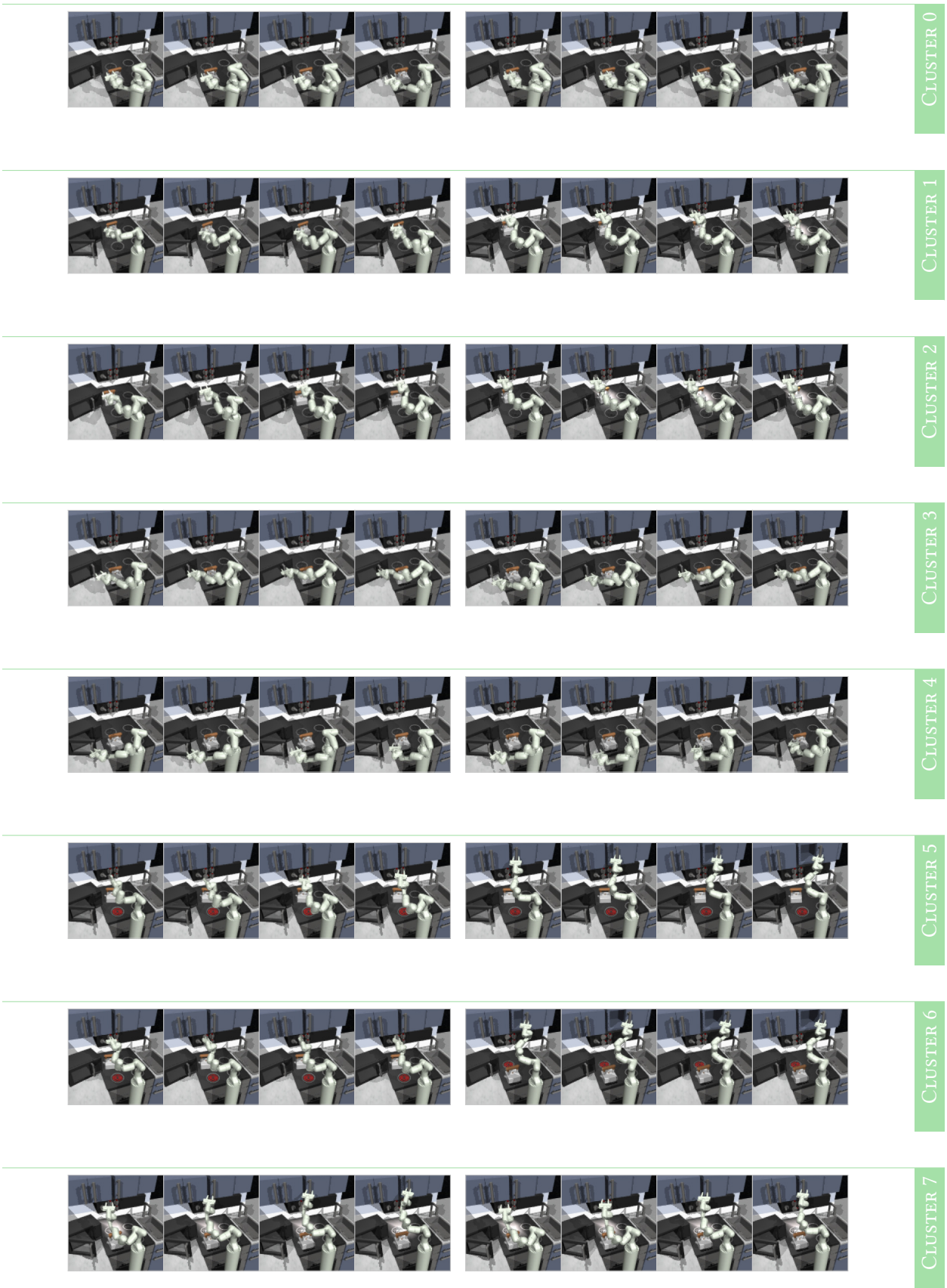


Figure B.7: Kitchen skill visualizations. We randomly sample 2 labeled skill trajectories (no cherry-picking) and visualize the trajectory's images in sequence after labeling with EXTRACT's skill extraction phase. Clusters are generally *semantically aligned.*

Figure B.8: LIBERO. We randomly sample 2 labeled skill trajectories (no cherry-picking) and visualize the trajectory's images in sequence after labeling with EXTRACT's skill extraction phase. Clusters are generally *semantically aligned*.

B.5 EXTRACT RL Performance Analysis

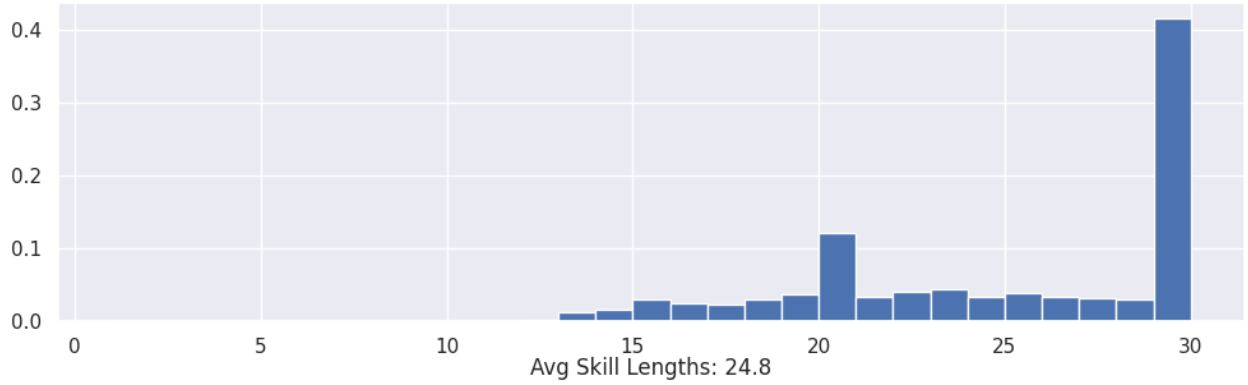


Figure B.9: Skill lengths histogram of actually used EXTRACT skills in Franka Kitchen at training convergence. As explained in Appendix B.2.1, we limit skill execution lengths to 30 in Franka Kitchen.

Our method's performance improvement over SPiRL is likely due to two reasons: longer average skills and a semantically structured skill-space instead of the random latent skills that SPiRL learns. In Section 4.5.3 we analyze the semantically structured skill-space. Here, we additionally analyze the longer average skills.

As plotted in Appendix Figure B.4, EXTRACT extracts skills of various lengths, many of which are quite long. This translates into longer-executed skills: we plot a histogram of the lengths of the skills the skill-based policy actually learns to use at convergence in Franka Kitchen in Figure B.9. EXTRACT-executed skills average 25 timesteps in length as compared to 10 for SPiRL. We experimented with longer skill lengths for SPiRL, but online RL performance suffered, a finding consistent with results presented in their paper [282].

Longer skills shorten the effective time horizon of the task by a factor of the average skill length for the skill-based agent because the skill-based agent operates on an MDP where transitions are defined by the end of execution of a *skill* which can be comprised of many low-level environment actions. By shortening the task time horizon, the learning efficiency of temporal-difference learning RL algorithms [339] can be

improved by, for example, reducing value function bootstrapping error accumulation as there are less timesteps between a sparse reward signal and the starting state.

B.6 Limitations

While EXTRACT enables efficient transfer learning, we still need the initial dataset from environments similar to the target environments for learning skills from. It would be useful to extend EXTRACT to data from *other robots* or *other environments* significantly different from the target environment to ease the data collection burden—possibly wth sim to real techniques [400]. Furthermore, in future work, we plan to combine our method with offline RL [107, 278, 187, 330, 97, 208] to learn skills from suboptimal data *without* the need to interact with an environment, targeting even greater sample efficiency. Furthermore, while EXTRACT is more sample-efficient than all other comparisons, it still requires many online samples to learn to execute new tasks with RL. We plan to investigate future directions that will allow us to combine offline learning approaches such as offline reinforcement learning with skill learning on the offline dataset to allow even more efficient transfer. Finally, EXTRACT requires image observations for the VLMs; skill learning from more input modalities would be interesting future work. Processing more input modalities to learn skills would be interesting future work.

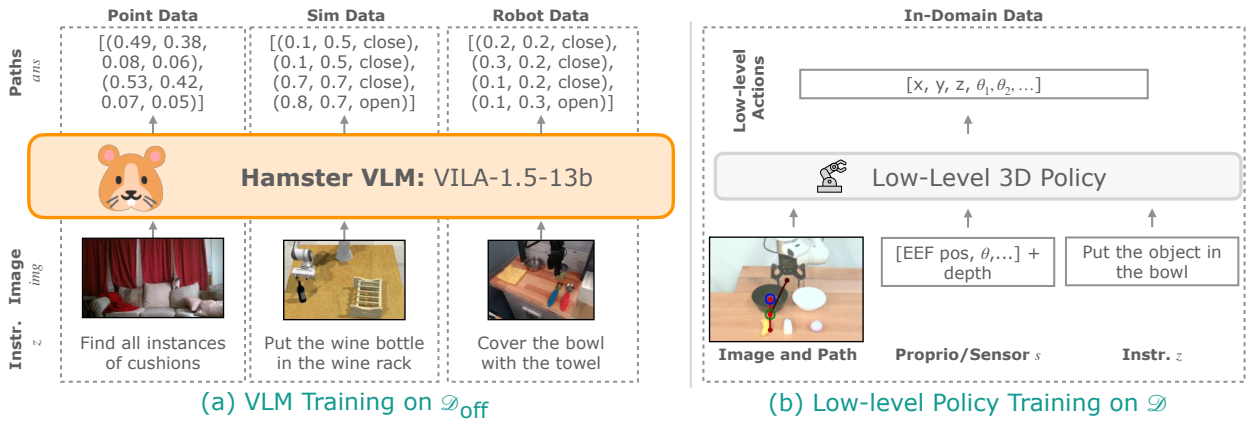
Figure C.1: Examples of various robot tasks and environments that HAMSTER can handle. See more details in our teaser video at https://hamster-robot.github.io/.

Appendix C

HAMSTER

C.1 VLM Finetuning Dataset Details

Pixel Point Pred Data. Our point prediction dataset comes from Robopoint [398]. 770k samples in our point prediction dataset contain labels given as a set of unordered points such as $p^o = [(0.25, 0.11), (0.22, 0.19), (0.53, 0.23)$ or bounding boxes in $[(cx, cy, w, h)]$ style. Other than that, following Robopoint [398], we use the VQA dataset [204] with 660k samples which answer VQA queries in natural language such as "What is the person feeding the cat?" We keep these data as is because these VQA queries are likely to benefit a VLM's



(a) VLM Training on $\mathcal{D}_{\text{off}}$ (b) Low-level Policy Training on $\mathcal{D}$

Figure C.2: (a): Examples of training data in $\mathcal{D}_{\text{off}}$ used to train HAMSTER's VLM. (b): The data used to train HAMSTER's low-level policies.

semantic reasoning and visual generalization capabilities; we fine-tune HAMSTER's VLM on the entire Robopoint dataset as given.

Simulation Data. We selected 81 RLBench tasks out of 103 to generate data by removing tasks with poor visibility on the `front_cam` view in RLBench. We use the first image in each episode combined with each language instruction. The final dataset contains around 320k trajectories.

Real Robot Data. For the Bridge [362] dataset, which only provides RGB images, we extract trajectories by iteratively estimating the extrinsic matrix for each episode. In each scene, we randomly sample a few frames and manually label the center of the gripper fingers. Using the corresponding end-effector poses, we compute the 3D-2D projection matrix with a PnP (Perspective-n-Point) approach. We then apply this projection matrix to the episodes and manually check for any misalignments between the projected gripper and the actual gripper. Episodes exhibiting significant deviations are filtered out, and a new round is started to estimate their extrinsic matrix.

For DROID [163], a large portion of the dataset contains noisy camera extrinsics information that do not result in good depth alignment. Therefore, we filter out trajectories with poor-quality extrinsics as measured by the alignment between the projected depth images and the RGB images. This results in ∼45k

trajectories (∼22k unique trajectories as trajectories each have 2 different camera viewpoints) which we use for constructing the VLM dataset $\mathcal{D}_{\text{off}}$ as described in Section 5.4.1.

C.2 Implementation and Architecture Details

HAMSTER Prompt

In the image, please execute the command described in ⟨quest⟩{quest}⟨/quest⟩.
Provide a sequence of points denoting the trajectory of a robot gripper to achieve the goal.
Format your answer as a list of tuples enclosed by ⟨ans⟩ and ⟨/ans⟩ tags. For example:
⟨ans⟩`[(0.25, 0.32), (0.32, 0.17), (0.13, 0.24),` ⟨action⟩`Open Gripper`⟨/action⟩`, (0.74, 0.21),` ⟨action⟩`Close Gripper`⟨/action⟩`, ...]`⟨/ans⟩
The tuple denotes the x and y location of the end effector of the gripper in the image. The action tags indicate the gripper action.
The coordinates should be floats ranging between 0 and 1, indicating the relative locations of the points in the image.

Figure C.3: The full text prompt we use to train HAMSTER with on simulation and real robot data (Section 5.4.1). We also use this prompt for inference.

C.2.1 VLM Implementation Details

VLM Prompt. We list the prompt for both fine-tuning on sim and real robot data and evaluation in Figure C.3. We condition the model on an image and the prompt, except when training on Pixel Point Prediction data (i.e., from Robopoint [398]) where we used the given prompts from the dataset. Note that we ask the model to output gripper changes as separate language tokens, i.e., `Open Gripper/Close Gripper`, as opposed to as a numerical value as shown in simplified depictions like Figure 5.2.

VLM Trajectory Processing. As mentioned in Section 5.4.1, one problem with directly training on the path labels p^o is that many paths may be extremely long. Therefore, we simplify the paths p^o with the Ramer-Douglas-Peucker algorithm [295, 83] that reduces curves composed of line segments to similar curves composed of fewer points. We run this algorithm on paths produced by simulation and real robot

data to generate the labels p^o for $\mathcal{D}_{\text{off}}$. We use tolerance $\epsilon = 0.05$, resulting in paths that are around 2-5 points for each short horizon task.

VLM Training Details. We train our VLM, VILA1.5-13B [198], on a node equipped with eight NVIDIA A100 GPUs, each utilizing approximately 65 GB of memory. The training process takes about 30 hours to complete. We use an effective batch size of 256 and a learning rate of 1×10^{-5}. During fine-tuning, the entire model—including the vision encoder—is updated.

C.2.2 Low-level Policy Training Details

We train RVT2 [110] and 3D-DA [162] as our lower-level policies. We keep overall architecture and training hyperparameters the same as paper settings. Specific details about how the inputs were modified other than the 2D path projection follow.

For low-level policy training, we train the policies on ground truth paths constructed by projecting trajectory end-effector points to the camera image. In order to also ensure the policies are robust to possible error introduced by HAMSTER VLM predictions during evaluation, we add a small amount of random noise ($N(0, 0.01)$) to the 2D path (x, y) image points during training to obtain slightly noisy path drawings. No noise was added to the gripper opening/closing indicator values.

RVT2 [110]. We remove the language instruction for RVT-2 when conditioning on HAMSTER 2D paths.

3D-DA [162]. In simulated experiments in Colosseum, no changes were needed. In fact, we saw a performance drop for HAMSTER+3D-DA when removing language for Colosseum tasks and a small drop in performance when using simplified language instructions. This is likely due to 3D-DA's visual attention mechanism which cross attends CLIP language token embeddings with CLIP visual features, therefore detailed language instructions are beneficial.

RT-Trajectory GPT-4o Prompt

In the image, please execute the command described in '{quest}'.
Provide a sequence of keypoints denoting a trajectory of a robot gripper to achieve the goal. Keep in mind these are keypoints, so you do not need to provide too many points.
Format your answer as a list of tuples enclosed by `<ans>` and `</ans>` tags. For example:
`<ans>[(0.25, 0.32), (0.32, 0.17), (0.13, 0.24), <action>Open Gripper</action>, (0.74, 0.21), <action>Close Gripper</action>, ...]</ans>`
The tuple denotes point x and y location of the end effector of the gripper in the image. The action tags indicate the gripper action.
The coordinates should be floats ranging between 0 and 1, indicating the relative locations of the points in the image.
The current position of the robot gripper is: {current_position}. Do not include this point in your answer.

Figure C.4: The full text prompt we use to prompt RT-Trajectory with GPT4-o.

In real-world experiments, we simplify the language instruction in the same way as for RVT2 when conditioning on HAMSTER 2D paths to encourage following the trajectory more closely with limited data. In addition, we reduced the embedding dimension of the transformer to 60 from 120, removed proprioception information from past timesteps, and reduced the number of transformer heads to 6 from 12 in order to prevent overfitting.

C.3 Real World Experiment Details

C.3.1 Training Tasks and Data Collection

For our real-world experiments, we collected all data using a Franka Panda arm through human teleoperation, following the setup described in Khazatsky et al. [163]. Below, we describe the training tasks:

Pick and place. We collected 220 episodes using 10 toy objects. In most of the training data, 2 bowls were placed closer to the robot base, while 3 objects were positioned nearer to the camera. The language goal for training consistently followed the format: `pick up the {object} and put it in the {container}`.

RT-Trajectory Code as Policies Prompt

Task Instruction: {task_instruction}
Robot Constraints:

- The robot arm takes as input 2D poses with gripper open/closing status of the form $(x, y, \text{gripper\_open} == 1)$
- The gripper can open and close with only binary values
- The workspace is a 1×1 square centered at $(0.5, 0.5)$
- The x-axis points rightward and y-axis points downward.

Please write Python code that generates a list of 2D poses and gripper statuses for the robot to follow. Include Python comments explaining each step. Assume you can use `numpy` or standard Python libraries, just make sure to import them.
Enclose the start and end of the code block with `<code>` and `</code>` so that it can be parsed. Make sure that it is a self-contained script such that when executing the code string, there is a variable named `robot_poses` which is a list of poses of the form: `[(x, y, gripper), (x, y, gripper), ...]`.
Scene Description:

```
<code>
{scene_description}
</code>
```

Figure C.5: The full text prompt we use for RT-Trajectory with Code-as-Policies on top of GPT4-o. The scene description at the bottom comes from an open-vocabulary object detector describing each detected object and its bounding box in the image based on the task instruction.

Knock down objects. We collected 50 episodes with various objects of different sizes. Typically, 3 objects were arranged in a row, and one was knocked down. The language goal for training followed the format: `push down the {object}`.

Press button. We collected 50 episodes with 4 colored buttons. In each episode, the gripper was teleoperated to press one of the buttons. The language goal followed the format: `press the {color} button`.

When training RVT2, which requires keyframes as labels, in addition to labeling frames where the gripper performs the `open gripper` and `close gripper` actions, we also included frames that capture the intermediate motion as the gripper moves toward these keyframes.

C.3.2 Baseline Training Details

OpenVLA [165]. Following Kim et al. [165], we only utilize parameter efficient fine-tuning (LoRA) for all of our experiments, since they showed that it matches full fine-tuning performance while being much more efficient. We follow the recommended default rank of r=32. We opt for the resolution of 360 x 360 to match all of the baseline model's resolutions. We also follow the recommended practice of training the model until it surpasses 95% token accuracy. However, for some fine-tuning datasets, token accuracy converged near 90%. We selected the model checkpoints when we observed that the token accuracy converged, which usually required 3,000 to 10,000 steps using a global batch size of either 16 or 32. Training was conducted with 1 or 2 A6000 gpus (which determined the global batch size of 16 or 32). Emprically, we observed that checkpoints that have converged showed very similar performance in the real world. For example, when we evaluate checkpoint that was trained for 3,000 steps and showed convergence, evaluating on a checkpoint trained for 5,000 steps of the same run resulted in a very similar performance.

RT-Trajectory [115]. We implement two versions of RT-Trajectory for the comparison in Table C.2. The first (0-shot GPT-4o) directly uses GPT-4o to generate 2D paths with a prompt very similar to the one we use for HAMSTER, displayed in Figure C.4.

The second version implements RT-Trajectory on top of a Code-as-Policies [194], as described in RT-Trajectory. We use OWLv2 [244] to perform open-vocabulary object detection on the image to generate a list of objects as the scene description and then prompt RT-Trajectory with the prompt shown in Figure C.5. We also use GPT-4o as the backbone for this method.

C.3.3 Evaluation Tasks

We evaluate our method on the tasks of `pick and place`, `knock down object`, and `press button` across various generalization challenges, as illustrated in Figure 5.4. Detailed results are available in **??**. Following [165], we assign points for each successful sub-action. For VLM, human experts are employed to assess the correctness of the predicted trajectories.

C.4 Extended Results

C.4.1 Impact of Design Decisions on VLM performance

To better understand the transfer and generalization performance of the proposed hierarchical VLA model, we analyze the impact of various decisions involved in training the high-level VLM. We conduct a human evaluation of different variants of a trained high-level VLM on a randomly collected dataset of real-world test images, as shown in Figure 5.7. We ask each model to generate 2D path traces corresponding to instructions such as "move the block on the right to Taylor Swift" or "screw the light bulb in the lamp" (the full set is in Appendix C.4.2). We then provide the paths generated by each method to human evaluators who have not previously seen any of the models' predictions. The human evaluators then rank the predictions for each method; we report the average rank across the samples in Table C.2.

Category	Task	OpenVLA	RVT2	RVT2+Sketch	3DDA	3DDA+Sketch
Basic	pick up the corn and put it in the black bowl	1	1	1	0	0.25
Basic	pick up the grape and put it in the white bowl	1	0.75	1	0	1
Basic	pick up the milk and put it in the white bowl	0	1	1	0	0.25
Basic	pick up the salt bottle and put it in the white bowl	0.75	0.5	1	0	0
Basic	pick up the shrimp and put it in the red bowl	0.75	0.5	1	0	1
Basic	pick up the cupcake and put it in the red bowl	0	0.5	0.5	0.25	1
Basic	press down the red button	0.5	0	1	0	1
Basic	press down the green button	0	1	0	0	0.25
Basic	press down the yellow button	0	0	1	0	1
Basic	press down the blue button	0.5	0	1	0	0.5
Basic	push down the green bottle	0.5	0	0.5	0	1
Basic	push down the pocky	0	1	1	0	0.5
Basic	push down the red bag	0.5	0.5	0	0	0.5
Basic	push down the bird toy	0	0	0	0	0.5
Basic	push down the yellow box	1	0	1	0	0.5
Object and Goal	pick up the salt bottle and put it in the white bowl	1	1	1	0.5	1
Object and Goal	pick up the banana and put it in the black bowl	0.25	0.25	1	0.5	1
Object and Goal	pick up the grape and put it in the black bowl	1	0.25	0.5	1	1
Object and Goal	pick up the carrot and put it in the red bowl	0.75	0	1	0.5	1
Object and Goal	pick up the milk and put it in the white bowl	0.25	0	1	0	0.25
Object and Goal	pick up the shrimp and put it in the white bowl	0.25	0.75	0.5	0.25	1
Object and Goal	pick up the cupcake and put it in the black bowl	0.25	0	1	0.5	0.75
Object and Goal	pick up the icecream and put it in the black bowl	0.25	0	0.5	0.5	1
Object and Goal	pick up the corn and put it in the red bowl	1	0	1	1	1
Object and Goal	pick up the green pepper and put it in the red bowl	0.75	0	0.5	0	0.25
Object and Goal	pick up the orange and put it in the white bowl	0.25	0	0	0	0
Visual(Table Texture)	pick up the salt bottle and put it in the white bowl	1	1	1	0	1
Visual(Table Texture)	pick up the banana and put it in the black bowl	0.25	0.25	0.75	0.5	0.75
Visual(lighting)	pick up the grape and put it in the black bowl	0.25	0	0.5	0.25	0
Visual(lighting)	pick up the carrot and put it in the red bowl	0.75	0	1	0	0.75
VIsual(clutter)	pick up the milk and put it in the white bowl	0.75	0.25	1	0.25	1
VIsual(clutter)	pick up the shrimp and put it in the red bowl	0.75	0.5	0	0	0.5
Visual(mix)	pick up the green pepper and put it in the red bowl	0.25	0	1	0	0.25
Visual(mix)	pick up the salt bottle and put it in the white bowl	0.25	0	0.25	0.25	1
Visual(appearance change)	pick up the green pepper and put it in the black bowl	1	0	0.5	0	1
Visual(appearance change)	pick up the salt bottle and put it in the black bowl	1	1	1	0	1
Visual(Table Texture)	press down the red button	1	1	0	0	0.5
Visual(lighting)	press down the green button	1	0	0.5	0	0.5
VIsual(clutter)	press down the yellow button	0	0	0.5	0	0.5
Visual(mix)	press down the blue button	0	0	0	0	0.5
Visual(Table Texture)	push down the pocky	0	1	0	0	0
VIsual(clutter)	push down the green bottle	1	0.5	1	0	1
VIsual(clutter)	push down the chocolate box	1	0	0	0	1
Visual(mix)	push down the green bottle	0	0	0.5	0	1
Language	pick up the sweet object and put it in the red bowl	1	1	1	0	1
Language	pick up the spicy object and put it in the red bowl	1	0	1	0	0.75
Language	pick up the salty object and put it in the red bowl	0	0	1	0	1
Language	pick up the object with color of cucumber and put it in the red bowl	0	0	1	0.25	0.75
Language	pick up the object with color of lavender and put it in the black bowl	0	0	1	0	1
Language	pick up the object with the color of sky and and put it in the container with the color of coal	1	0	0	0.25	1
Language	pick up the block with the color of sunflower and put it in the container with the color of enthusiasm	0	0.25	1	0	1
Language	press the button with the color of fire	0.5	0	1	0	0.5
Language	press the button with the color of cucumber	0	0	1	0	0.5
Language	press the button with the color of sky	0	0	0	0.5	1
Language	press the button with the color of banana	0	0	0	0	0.5
Language	push down the object with color of leaf	0	1	1	0	0
Language	push down the box contains cruchy biscuit	0	0	0	0	1
Language	push down the bag with color of fire	0	0	1	0	0.5
Language	push down the object with feather	0.5	0	1	0	1
Spatial	pick up the left object and put it in the left bowl	0	1	1	0.25	1
Spatial	pick up the middle object and put it in the left bowl	0	0	1	0	1
Spatial	pick up the right object and put it in the left bowl	1	0	0.5	0.25	0.5
Spatial	pick up the left object and put it in the right bowl	0.25	0.25	1	0.25	1
Spatial	pick up the middle object and put it in the right bowl	0	0	1	0	1
Spatial	pick up the right object and put it in the right bowl	0.5	0	1	0	1
Spatial	press down the left button	0.5	0	0	0	0.5
Spatial	press down the middle button	0	0	1	1	0.5
Spatial	press down the right button	0	0	1	1	1
Spatial	push down the left object	0.5	0	0	0	0
Spatial	push down the middle object	1	0.5	0	0	1
Spatial	push down the right object	0.5	0	0.5	0.5	1
Novel Object	pick up the "R" and put it in the red bowl	0	0	1	0	1
Novel Object	pick up the boxed juice and put it in the red bowl	0	0.75	0.75	1	1
Novel Object	pick up the cholate bar and put it in the white bowl	0.25	0	0.5	0.5	1
Novel Object	pick up the smile face and put it in the red bowl	1	0	1	0	1
Novel Object	pick up the mouse and put it in the red bowl	0	0.25	1	0	1
Novel Object	pick up the 5 and put it in the white bowl	0	0	0	0	0.25
Multiple	pick up the lays chip and put it in the pan	0.25	0.25	0.75	0	1
Multiple	pick up the garlic and put it in then pan	0.25	0	1	0	0.25
Multiple	pick up the "K" and put it in the pan	0.25	0	0.5	0	1
Multiple	pick up the pocky and put it in the pan	0	0.25	0	0.25	0.25

Table C.1: Detailed results of real-world evaluation. The first column indicates the variation category, while the second column presents the language instruction. For the `pick and place` task, we report the success rate and the number of successful executions. For the `press button` and `knock down` tasks, we only report the success rate.

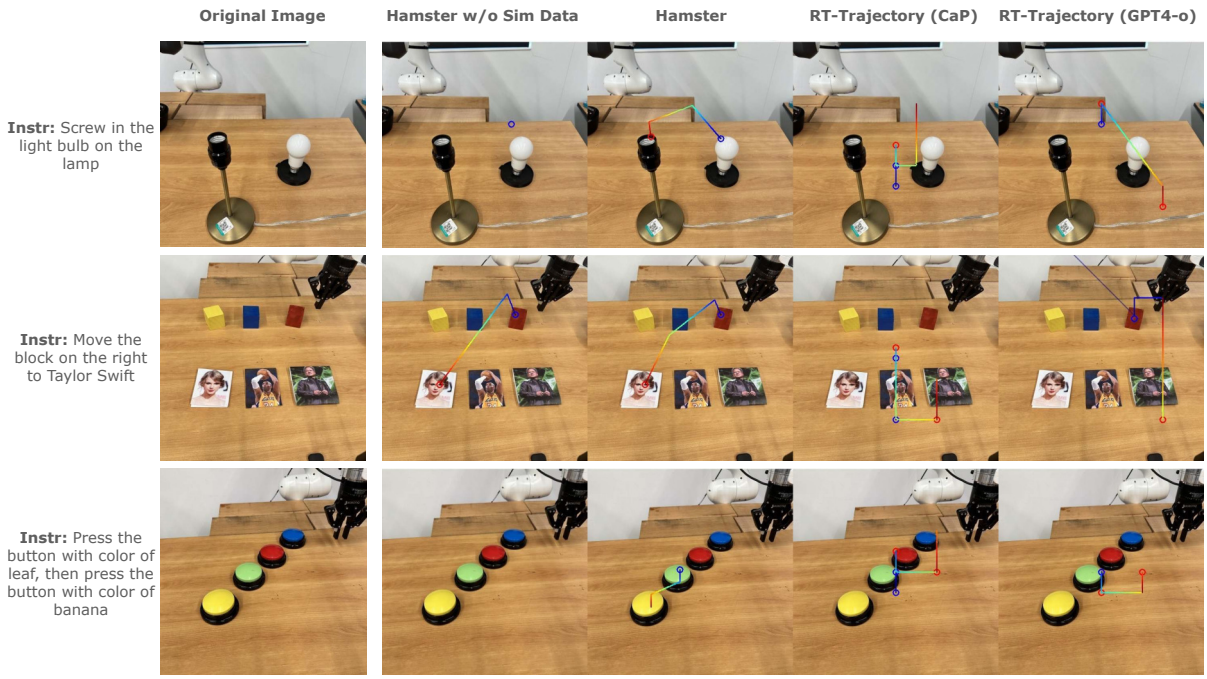


Figure C.6: Human VLM evaluation example images and instructions along with corresponding trajectories from HAMSTER without any finetuning on (RLBench) simulation data, HAMSTER finetuned on simulation data, and GPT-4o.

We evaluate the following VLM models: (1) zero-shot state-of-the-art closed-source models such as GPT-4o using a similar prompt to ours (shown in Figure C.4), (2) zero-shot state-of-the-art closed-source models such as GPT-4o but using Code-as-Policies [194] to generate paths as described in Gu et al. [115] (prompt in Figure C.5), (3) finetuned open-source models (VILA-1.5-13b) on the data sources described in Section 5.4.1, but excluding the simulation trajectories from the RLBench dataset, (4) finetuned open-source models (VILA-1.5-13b) on the data sources described in Section 5.4.1, including path sketches from the RL-Bench dataset. The purpose of these evaluations is to first compare with closely related work that generates 2D trajectories using pretrained closed source VLMs [115] (Comparison (1) and (2)). The comparison between (3) and (4) (our complete method) is meant to isolate the impact of including the simulation path sketches from the RLBench dataset. In doing so, we analyze the ability of the VLM to predict intermediate paths to transfer across significantly varying domains (from RLBench to the real world).

Method	VLM	Finetuning Data	Rank Exc. Real RLB.	Rank Real RLB.	Rank All
RT-Traj.	0-shot GPT-4o	-	3.40	3.63	3.47
RT-Traj.	CaP GPT-4o	-	3.57	3.36	3.41
HAMSTER	VILA	Our Exc. Sim RLB.	1.78	2.39	2.13
HAMSTER	VILA	Our	**1.59**	**1.28**	**1.40**

Table C.2: Ranking-based human evaluation of different VLMs, averaged across various real-world evaluation tasks. Results indicate that HAMSTER including simulation data is most effective since it captures both spatial and semantic information across diverse tasks from RLBench. This significantly outperforms zero-shot VLM-based trajectory generation, as described in Gu et al. [115]

The results suggest that: (1) zero-shot path generation, even from closed-source VLMs [115] such as GPT-4o with additional help through Code-as-Policies [194], underperforms VLMs finetuned on cross-domain data as in HAMSTER; (2) inclusion of significantly different training data such as low-fidelity simulation during finetuning improves the real-world performance of the VLM. This highlights the transferability displayed by HAMSTER across widely varying domains. These results emphasize that the hierarchical VLA approach described in HAMSTER can effectively utilize diverse sources of cheap prior data for 2D path predictions, despite considerable perceptual differences.

C.4.2 VLM Real World Generalization Study

The full list of task descriptions for this study is below (see Appendix C.4.1 for the main experiment details). Duplicates indicate different images for the same task. We plot some additional comparison examples in Figure C.6. Note that the path drawing convention in images for this experiment differ from what is given to the lower-level policies as described in Section 5.4.2 as this multi-colored line is easier for human evaluators to see.

1. screw in the light bulb on the lamp
2. screw in the light bulb on the lamp
3. screw in the light bulb on the lamp

4. screw out the light bulb and place it on the holder
5. screw out the light bulb and place it on the holder
6. screw in the light bulb
7. screw in the light bulb on the lamp
8. move the blue block on Taylor Swift
9. pick up the left block and put it on Jensen Huang
10. move the block on the right to Taylor Swift
11. place the yellow block on Kobe
12. pick up the blue block and place it on Jensen Huang
13. move the red block to Kobe
14. press the button on the wall
15. press the button to open the left door
16. press the button to open the right door
17. open the middle drawer
18. open the bottom drawer
19. open the top drawer
20. open the middle drawer
21. open the bottom drawer
22. press the button

23. press the button
24. press the orange button
25. press the orange button with black base
26. press the button
27. pick up the SPAM and put it into the drawer
28. pick up the orange juice and put it behind the red box
29. pick up the tomato soup and put it into the drawer
30. pick up the peach and put it into the drawer
31. move the mayo to the drawer
32. move the dessert to the drawer
33. pick up the object on the left and place it on the left
34. pick up the fruit on the left and put it on the plate
35. pick up the milk and put it on the plate
36. press the button with the color of cucumber, then press the button with color of fire
37. press the button with color of banana
38. press the button with color of leaf
39. press the button with color of leaf, then press the one with color of banana
40. press left button
41. pick up the left block on the bottom and stack it on the middle block on top

42. make I on top of C

43. put number 2 over number 5

44. stack block with lion over block with earth

45. pick up the left block on the bottom and stack it on the middle block on top

46. stack the leftest block on the rightest block

47. stack the block 25 over block L

48. put the left block on first stair

C.4.3 Human Ranking

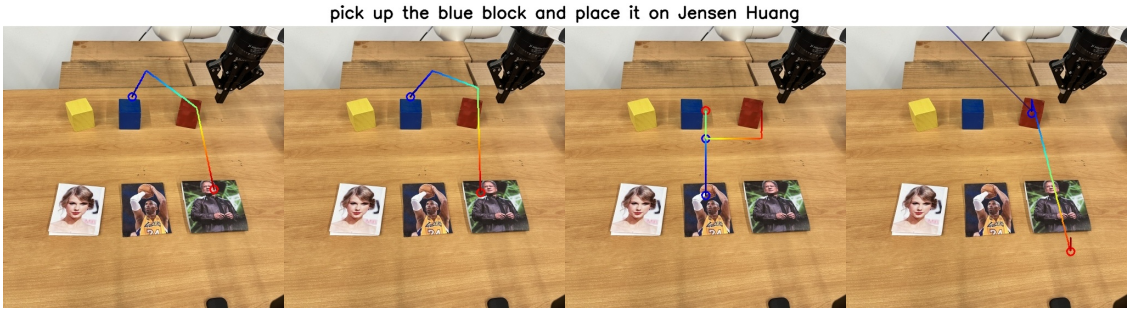


Figure C.7: An example of results for human ranking. The trajectory is from blue to red with blue circle and red circle denotes gripper close point and open point respectively. The grader is asked to provide a rank to these trajectory about which trajectory has highest chance to succeed.

Due to the variety of possible trajectories that accomplish the same task, we use human rankings to compare how likely produced trajectories are to solve the task instead of quantitative metrics such as MSE. To do that, we generate trajectories for 48 image-question pairs with HAMSTER w/o RLBench, HAMSTER, Code-as-Policy [194], and GPT4o [3]. See Figure C.7 for an example.

We recruit 5 human evaluators, who are robot learning researchers that have not seen the path outputs of HAMSTER, to grade these 4 VLMs based on the instruction: *"Provide a rank for each method (1 for best and 4 for worst). In your opinion, which robot trajectory is most likely to succeed. Traj goes from blue to red,*

blue circle means close gripper, red circle means open gripper." The evaluators are allowed to give multiple trajectories the same score if they believe those trajectories are tied. As they are robot learning researchers, they are familiar with the types of trajectories that are more likely to succeed. Therefore, these rankings act as a meaningful trajectory quality metric.

C.5 Failure Analysis

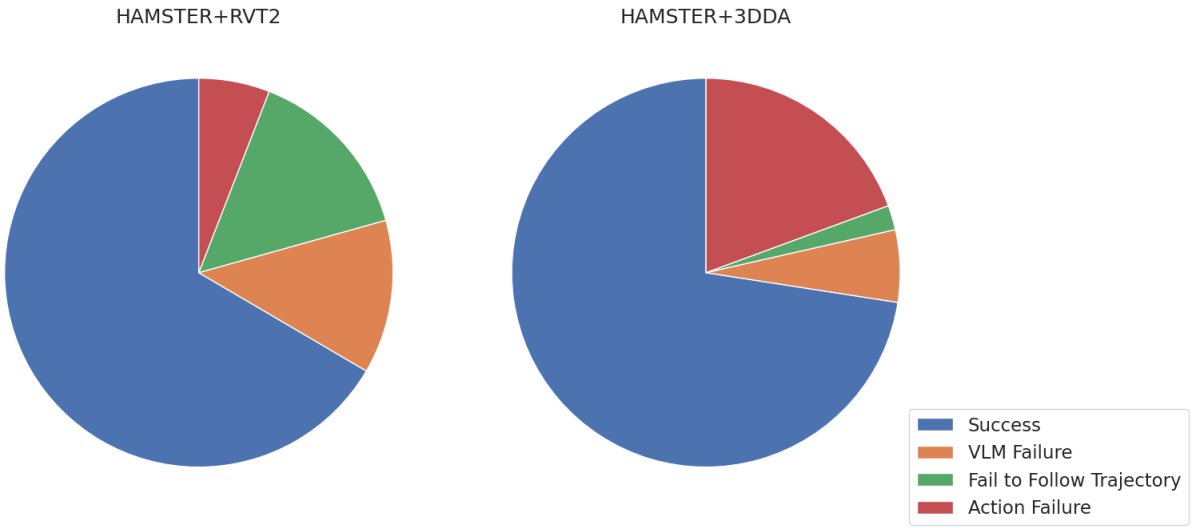


Figure C.8: Performance Distribution of RVT2+Sketch and 3DDA+Sketch

This section outlines the failure modes observed during our experiments and provides a detailed breakdown of the causes. Failures can be attributed to issues in **trajectory prediction**, **trajectory adherence**, and **action execution**.

C.5.1 Different Failure Modes

Trajectory Prediction Failures The Vision-Language Model (VLM) may fail to predict the correct trajectory due to several factors:

- *Failure to understand the language goal:* Although the VLM demonstrates strong capabilities in handling diverse task descriptions, it struggles when the training set lacks similar tasks. This can cause the model to misunderstand the goal and make inaccurate predictions.

- *Incorrect trajectory prediction:* In some cases, the VLM predicts an incorrect trajectory, either by interacting with the wrong objects or misinterpreting the direction of the affordance.

- *Dynamic changes in the environment:* Since trajectories are generated at the beginning of a task, significant environmental changes during execution can lead to failure. The model lacks the ability to dynamically adjust the trajectory or reidentify the object initially referenced.

Trajectory Adherence Failures Failures in adhering to the predicted trajectory arise primarily due to:

- *3D ambiguity:* The use of 2D trajectory predictions introduces ambiguities, such as determining whether a point is positioned above or behind an object, leading to execution errors.

- *Incorrect object interaction:* The low-level action model is not explicitly constrained to strictly follow the predicted trajectory. As a result, it may deviate, interacting with the wrong object and causing task failures.

Action Execution Failures Even when the trajectory is correctly predicted and adhered to, action execution may still fail due to:

- *Execution-specific issues:* Despite training on a diverse set of actions, the model may fail during execution. For example, in grasping tasks, an incorrect grasp angle can cause the object to slip, resulting in a failed grasp.

C.5.2 Failure Analysis

Our analysis in Figure C.8 reveals distinct failure tendencies across methods.

For RVT, 72% of failures stemmed from the low-level model failing to follow the trajectory, while 28% were due to execution failures. In contrast, for 3DDA, only 10% of failures were related to trajectory adherence, with 90% attributed to execution failures.

We hypothesize that this discrepancy arises because RVT incorporates a re-projection step, complicating trajectory adherence. In contrast, 3DDA leverages a vision tower that processes the original 2D image, simplifying trajectory interpretation.

C.6 Simulation Experiment Details

Our simulation experiments are performed on Colosseum [288], a simulator built upon RLBench [149] containing a large number of visual and task variations to test the generalization performance of robot manipulation policies (see Figure C.9 for a visualization of a subset of the variations). We use the `front_camera` and remove all tasks in which the camera does not provide a clear view of the objects in the task, resulting in 14 out of 20 colosseum tasks (we remove `basketball_in_hoop`, `empty_drawer`, `get_ice_from_fridge`, `move_hanger`, `open_drawer`, `turn_oven_on`).

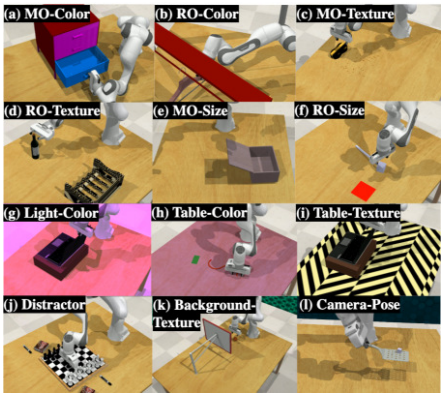


Figure C.9: Colosseum benchmark variations. Figure from Pumacay et al. [288], taken with permission.

Colosseum contains 100 training episodes for each task, without any visual variations, and evaluates on 25 evaluation episodes for each variation. We follow the same procedure other than using just the `front_camera` instead of multiple cameras. We report results in Table 5.3 after removing variations with no visual variations (e.g., object friction).

Task	RVT2	3DDA	OpenVLA	HAMSTER+RVT2	HAMSTER+3DDA
pick and place	0.28	0.19	0.46	0.79	0.78
press button	0.13	0.16	0.25	0.50	0.63
knock down	0.17	0.03	0.41	0.47	0.66

Table C.3: Real world average success rates grouped by task type.

C.7 Different ways of representing 2D Paths

To investigate the effect of the number of points on the 2D path, we train the VLM to predict 1. paths simplified using RDP algorithm, which simplify paths in short horizon tasks to 3-5 points and is what we used in the paper. We denote these paths as RDP in the following; 2. Paths represented with 20 points sampled on the path with same step size, denoted as 20p in the following. We keep points where the gripper is executing operation of open or close in both methods.

We train the network on RLBench 80 tasks with 1000 episodes for each task and test it on 25 episodes on the task of close jar. We tried both VILA1.5-3B (denoted as 3B) and VILA1.5-13B (denoted as 13B) as our backbone. Thus we have in total 4 combinations over 2 backbones and 2 designs of path representations. We visualize the result in this Figure C.10.

From this result we can see that when using smaller models, like VILA1.5-3B, paths represented using points extracted using RDP algorithm outperforms paths represented with a fixed number of 20 points significantly. When the network becomes larger to the level of 13B, the VLM is able to handle the representation using 20 points and both two path representations work perfectly. We believe that is because when points are simplified using the RDP algorithm, we usually need less points to represent the path and helps the model to pay more attention to predict the accurate position for the gripper open/close points.

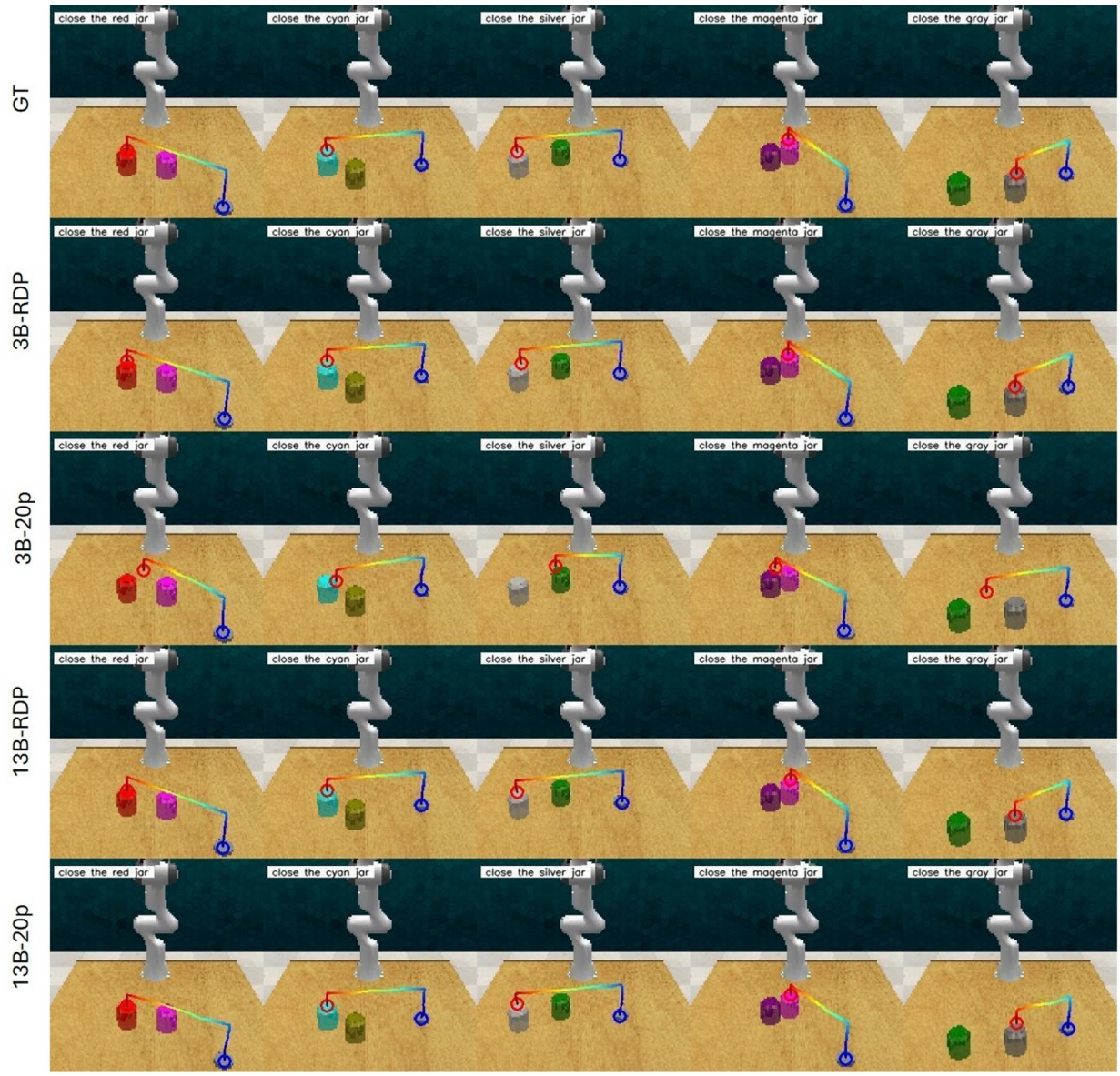


Figure C.10: The task is to pick up the lid and close it on the jar with correct color. Task description is located on the top-left corner of each image. The trajectory goes from blue to red where blue circles denotes where the gripper should close and red circles denotes where the gripper should open. GT denotes ground truth, 3B and 13B denotes VILA1.5-3B and VILA1.5-13B, RDP denotes paths simplified using Ramer–Douglas–Peucker algorithm while 20p denotes paths reprensented using 20 points.

Appendix D

TAIL

Appendix: TAIL: Task-specific adapters for imitation learning with large pretrained models

D.1 Model Architecture Details

D.1.1 Pretrained Input Encoders

We utilize pretrained CLIP image and textual encoders [289] to encode image observations and language goal descriptions, respectively. Note that we do not use a pre-trained encoder for the low-dimensional state; the state encoder is learned from scratch.

D.1.2 Input Modality Fusion

We utilize Feature-wise Linear Modulation (FiLM) layers [280] (Fig. 6.1(a), input fusion module) to fuse language task specifications with image observations. FiLM is a technique in multi-modal deep learning which modulates the intermediate activations of a neural network based on external information. Rather than explicitly designing architectures for conditional computation, FiLM layers simply use the features from one network to modulate the features of another.

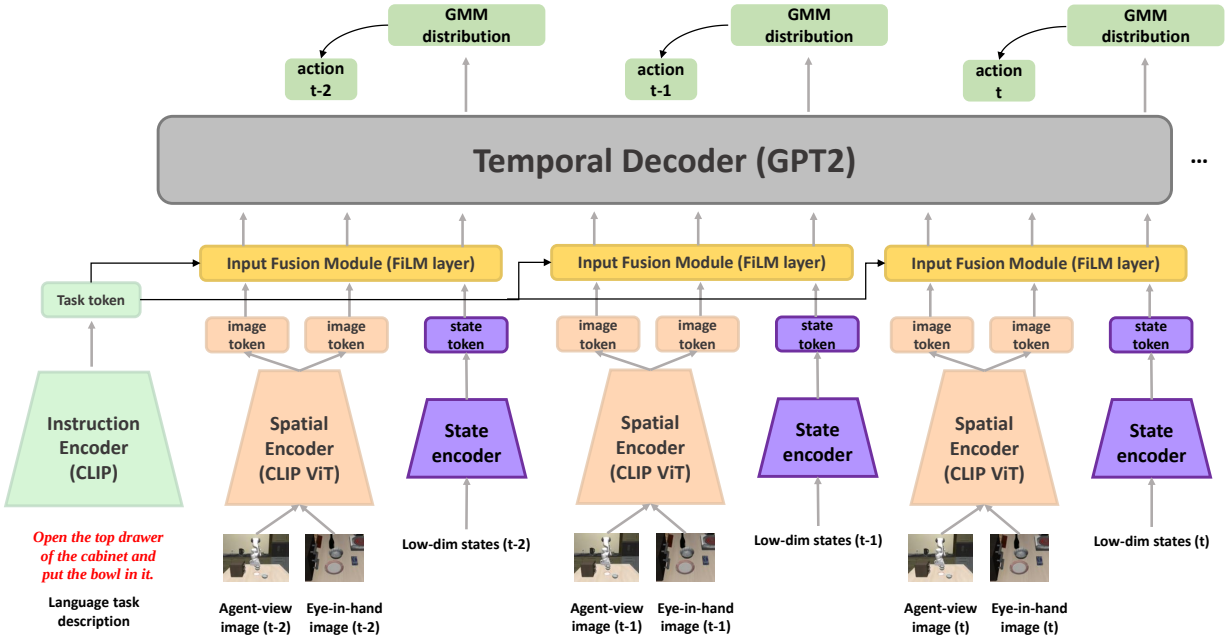


Figure D.1: A detailed view of the multi-modal, transformer policy architecture we utilize for pretraining. We encode language task descriptions with a pretrained CLIP instruction encoder and image observations with a pretrained CLIP spatial encoder. We additionally encode robot state observations which, along with the observation embeddings, are embedded into a sequence of tokens used by the temporal decoder transformer to predict single-step action distributions. We include an input fusion module (FiLM [280]) to explicitly combine the task embedding with the observation and state embeddings for better instruction-following ability.

Let's consider a neural network f with intermediate activations x and an external network g which outputs modulation parameters γ and β. The modulated features x' are given by:

$$\gamma, \beta = g(z) \tag{D.1}$$

$$x' = \gamma \odot x + \beta, \tag{D.2}$$

where z is the input to the external network g; $\odot$ represents element-wise multiplication; γ and β are vectors having the same size as x, with each element modulating a corresponding feature in x.

Thus, FiLM layers allow for a dynamic and feature-wise conditional computation without needing explicit architectural changes. As such, task token (language) embeddings are given as input to a fully connected feedforward network, which outputs scale and translation parameters for the image and state embeddings. These parameters modulate the image and state embeddings before they are passed to the transformer backbone.

D.1.3 Temporal Transformer Backbone

We utilize a standard GPT-2 [290] transformer backbone for our policy. Its input is a sequence of image and low-dim state encodings (robot joint states in LIBERO) and it outputs an action distribution. Following the literature [234, 201], we adopt a stochastic policy parametrization based on a Gaussian-Mixture-Model (GMM) [28]. Therefore, for every decision-making step, the transformer produces a latent vector of Gaussian means and variances, one for each of the GMM modes. We optimize the parameters of the model with the negative log-likelihood loss on the ground truth actions based on the parameters of the GMM distribution. At evaluation time, we deterministically select the next action parameterized by the mean of the Gaussian model with the highest density.

The environment configuration and the temporal decoder (GPT-2) hyperparameters are presented in Table D.1.

Table D.1: Environment configuration and GPT-2 model hyperparameters

Environment Configuration	
Action Dim.	7
Raw State Dim.	9
Max Episode Length	500
Image Resolution	128 x 128
Image Views	Agent Front, Eye-in-Hand

GPT2 Temporal Encoder Configuration			
Max Seq Length	8	Activation	Gelu New
Number of Heads	8	Number of Layers	6
GMM Min Std	0.0001	GMM Modes	5
FiLM Layers	2	Dropout	0.15

D.2 Implementation and Training Details

D.2.1 Baseline Details

Experience Replay (ER). ER [48, 301] is a rehearsal-based approach that retains a buffer of samples from previous tasks to facilitate the learning of new tasks. After completing the learning process for a task, a subset of the data is saved into this buffer. During the training of subsequent tasks, ER draws samples from this buffer and mixes them with current task data. This process ensures that the training data closely resembles the distribution of data across all tasks. In our setup, we store all the previous trajectories in a replay buffer. For each training iteration on a new task, we uniformly sample 50% trajectories from this buffer and 50% from the new task's training data, respectively.

Elastic Weight Consolidation (EWC). EWC [169] is a regularization method that adds a term to the standard single-task learning objective to constrain the updates of the neural network. This constraint uses the Fisher information matrix to gauge the significance of each network parameter. The loss function for task k is represented as:

$$L_{\text{EWC}_k}(\theta) = L_{\text{BC}_K}(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{k-1,i})^2$$

Here, λ is a hyperparameter penalty, and F_i is the diagonal of the Fisher information matrix given by:

$$F_k = \mathbb{E}_{s\sim D_k, a\sim p_\theta(\cdot|s)} \left(\nabla_{\theta_k} \log p_{\theta_k}(a|s)\right)^2$$

For our experiments, we adopt the online version of EWC. It updates the Fisher information matrix using an exponential moving average throughout the lifelong learning process. The actual Fisher Information Matrix estimate used is:

$$\tilde{F}_k = \gamma F_{k-1} + (1-\gamma) F_k$$

with $F_k = \mathbb{E}_{(s,a)\sim D_k} \left(\nabla_{\theta_k} \log p_{\theta_k}(a|s)\right)^2$ and k representing the task number. Following the benchmark implementation [201], the hyperparameters are set as $\gamma = 0.9$ and $\lambda = 5 \times 10^4$.

Discussions. Both Experience Replay (ER) and Elastic Weight Consolidation (EWC) demonstrate potential in mitigating catastrophic forgetting. However, they each come with notable limitations, particularly with respect to forward transfer performance, storage, and computational efficiency.

Storage Overhead: ER demands significant storage space to maintain samples from prior tasks. This becomes particularly evident when comparing the storage needs of ER for larger datasets, such as the Kitchen dataset which requires 28GB, with the lightweight LoRA adapter occupies only 7.8MB. The vast difference in storage demands underscores the inefficiency of the ER approach.

Computational Challenges: EWC, by design, necessitates the maintenance of a copy of the weights of the previous model in GPU memory. This leads to escalated GPU memory consumption, making EWC tends to reduce the training batch size, subsequently slowing down the training process.

Training Instability: The regularization approach of EWC can introduce instability during training, owing to the regularization loss. This is also reflected by the poor forward transfer capability, as shown in Table 6.1.

Scalability Concerns: While EWC might be manageable for smaller networks, it is ill-suited for the fine-tuning of larger decision models due to its computational and storage challenges.

Given these outlined limitations, we advocate TAIL for alternative approaches that are both storage-efficient and computationally scalable, especially for large pretrained model adaptation.

D.2.2 TAIL Adapter Configurations

To establish our TAIL adapter configurations, we primarily draw from the AdapterHub implementation, setup and hyperparameters [285].

We utilize the default hyperparameters for LoRA, with the rank $r = 8$ and scaling factor $\alpha = 8$. These low-rank matrices are applied in parallel to the Transformer's query and value matrices [141]. We also adopt the default for prefix token length of 30 for the prefix tuning [190] method across all tasks. To improve the training stability, Low-rank matrices ($r = 16$) are employed during training to represent the prefix tokens. The Bottleneck Adapter [140] employs the bottleneck size of 32, and is applied to both the output layer of the attention and the intermediate feedforward layers. The RoboAdapter method [322], as the closest work to us, also applies the sequential adapters to the decision-making domain. It differs from the Bottleneck Adapter in that they adopt a special insertion of weights to specific layers of the Transformer, namely, layers $0, 1, 5, 6, 10, 11$. They selectively skip certain layers, aiming to increase the bottleneck size on the remaining layers. Therefore, the bottleneck size is doubled to 64 for this approach, such that all methods share similar amount of parameters.

In order to maintain balanced adapter parameters number between the two CLIP-based (spatial and instruction) encoders, and the temporal transformer GPT2 decoder, the rank size for the GPT2 decoder is doubled across all methodologies. This adjustment compensates for the GPT2 decoder's fewer layers relative to the encoders.

For the continual learning setup, we use the previous stage's adapter weight (if any) plus a small random Gaussian noise with standard deviation 0.001 as an initialization of the current stage. The goal for adding a minor random noise aims to improve the adapter weight capacity [173, 5, 215], preventing the weights from being trapped into local optimum. There is a potential to better utilize the trained adapter weights on preceding tasks. We outline several promising exploration directions in Appendix Section D.2.4.

D.2.3 Training Hyperparameters and Experiment Configurations

Following similar setup as in the LIBERO benchmark [201], we perform data augmentation for the image observation data for all methods. We adopt the color, affine, and random erase augmentations to improve the robustness. The hyperparameters are presented in Table D.2.

Table D.2: Image data augmentation and training hyperparameters

Image Augmentation			
Brightness	0.3	Contrast	0.3
Saturation	0.3	Hue	0.3
Color Aug Prob.	0.9	Affine Degrees	15
Affine Translate	0.1	Affine Prob.	0.6
Random Erase Prob.	0.1		

Training and Optimizer Configuration			
Training Epochs	100/50	Batch Size (per device)	10/14/18
Training Epochs per Eval	5	Eval Episodes/Task	8
Warm-up Steps	500	Weight Decay	0.1
Learning Rate (LR)	1e-4	LR Scheduler	Linear
Training Demo Num	40	Validation Demo Num	40

For our training process, we employed the AdamW optimizer combined with a linear learning rate scheduler. The majority of our task suites—Kitchen, Spatial, Goal, Object, Living Room, and Study Room—underwent training for 100 epochs. Notably, each suite encompasses multiple tasks, with Kitchen having 40 and the others containing 8 each. In contrast, the 10 long-horizon adaptation tasks, termed LIBERO-10, were trained for 50 epochs, with each task trained sequentially. We performed evaluations after every 5 training epochs over 8 episodes (unseen in training) for each task.

Computing machine. Our experimental platform was powered by an AMD EPYC 7R32 CPU running Ubuntu 20.04.06. All trainings utilized 8 NVIDIA A10G GPUs, each with a memory of 22731 MiB, equipped with driver version 470.199.02 and CUDA version 11.4. We employ Distributed Data Parallel (DDP) for parallel training across 8 GPUs, and utilize the 16-bit floating point precision (FP16) training mode to accelerate the training process. To ensure reproducibility, we adopted 3 distinct random seeds: 0, 21, and 42.

Training time. For a holistic perspective on training duration: FFT and ER methods demanded between $120 \sim 140$ hours per experiment ($1.5 \sim 1.75$ hours per task) for the 6 task suites shown in Fig. 6.5, including the evaluation time. In stark contrast, TAIL-based techniques slashed this to $60 \sim 66$ hours

($0.75 \sim 0.825$ hours per task). Hence, TAIL would also be much cheaper to train, considering its significantly shorter training time under identical computing machines.

Batch sizes varied by training method. EWC employed a batch size of 10, given its added memory demands to store a distinct full parameter set. FFT and ER utilized batch sizes of 14. Owing to TAIL's more efficient memory utilization—detailed in Table 6.3—a larger batch size of 18 was feasible, which can maximize GPU resource usage on our machine, reducing training duration and cost.

D.2.4 More Discussion and Future Directions

The TAIL framework paves the way for a myriad of research opportunities:

1. **Better Weight Allocation Method Across Layers:** An interesting question within this framework is discerning which layers, early or later, derive the most benefit from weight modifications. This can offer insights into the adaptability of neural architectures [182].

2. **Enhanced Reusability of Trained Adapters:** Exploring methods to efficiently reuse adapters from prior tasks, especially in scenarios with limited data, is a promising direction. AdapterFusion techniques [284] can be potentially useful, enabling the composition of knowledge from multiple pre-existing adapters.

3. **Building on Knowledge with Parallel Integration:** The parallel integration method, particularly with LoRA weights, offers the capability to merge trained weights back into the main model. This iterative buildup of knowledge makes the approach valuable for continual learning, allowing new adapters to capitalize on the expertise of their predecessors.

4. **Combining with Established Continual Learning Strategies:** The potential to merge the TAIL framework with existing continual learning methods, like Experience Replay and EWC, can be a beneficial avenue. Such integrations can accommodate the strengths of each method, crafting models that are both efficient in memory and robust against forgetting.

5. **Extension beyond the Imitation Learning Domain:** Taking the TAILframework into other decision-making domains, such as reinforcement learning (RL), is also promising. TAIL has the potential to address the model capacity loss issue in RL [5, 215]. Leveraging the TAIL framework can also aid in multitask learning, meta-learning, and efficiently adapting offline-trained RL models to new tasks without the necessity of vast amounts of data or extensive fine-tuning, thereby potentially accelerating convergence to optimal policies.

The avenues above elucidate the adaptability and potential of the TAIL framework, setting the stage for future research in this domain.

D.3 More Experiment Results

D.3.1 Overfitting

For each task, we used 40 demonstrations for training and 10 for validation. We are interested in the following question: *In scenarios where data is limited, how resilient is TAIL against overfitting compared to traditional fine-tuning methods?* To answer this, we present the training and validation loss cross the Kitchen, Spatial, Goal, Object, Living Room and Study Room task suites, each with 100 epochs, in Fig. D.2.

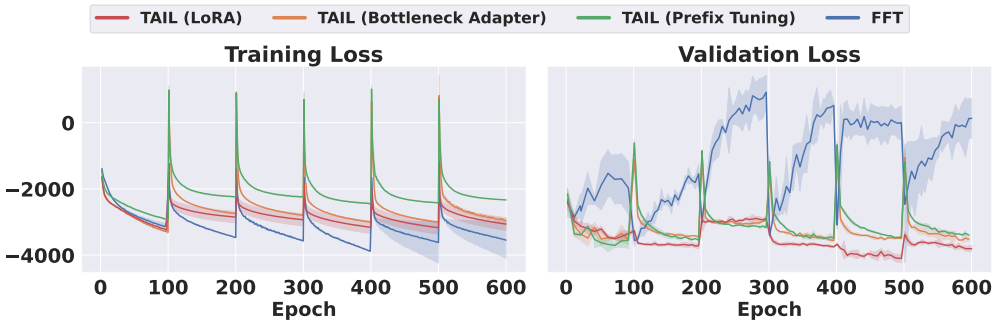


Figure D.2: Adaptation loss trends: Training versus validation. The graph shows that the TAIL model consistently has more stable validation losses, which means that it is more robust to contexts with limited data. On the other hand, the full fine-tuning model (FFT) has larger validation losses, which means that it is more likely to overfit to the training data.

A noteworthy observation from Fig. D.2 is the behavior of FFT. Despite achieving the lowest training loss across all stages, its validation loss spikes significantly after just a few epochs. This pattern suggests severe overfitting when FFT is applied to the entire parameter space using limited data. Intriguingly, this overfitting intensifies in the later adaptation phases, potentially signifying a distortion of pretrained features as alluded to by Kumar et al. [173]. Such distortion could be a contributor to the suboptimal success rate observed in Fig. 6.5, and the loss of learning capacity when revisiting a previous task, as presented in Table 6.2.

In constrast, TAIL-based methods shows strong resilience against overfitting. Drawing from the Occam's razor principle, TAIL leverages fewer trainable parameters, inherently reducing its potential to overfit with scarce data. Additional, different integration styles provide the flexibility to effectively utilize the features from pretrained models while preserving them across all the adaptation stages.

This observation underscores the disparities between our decision-making problem, characterized by its limited data, and the traditional language or vision domains, which have data in abundance. Prior studies utilizing parameter-efficient fine-tuning techniques for language or vision tasks often reported superior performance with full fine-tuning due to its low training loss [132, 235, 55, 322]. However, as our results demonstrate, a lower training loss does not invariably translate to superior performance, especially in the context of a data-scarce sequential decision-making tasks.

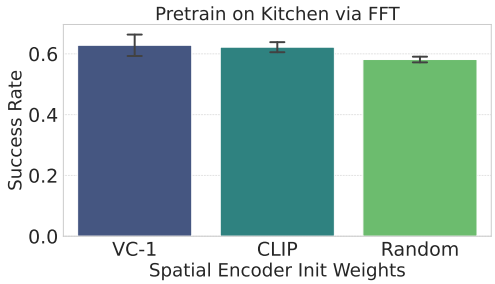


Figure D.3: Training on the Kitchen task with different pretrained CLIP-ViT encoder weight. Random means using random initialization weight.

D.3.2 Analysis of pretrained weights' influence

We aim to answer the following question: *how does the underlying pretrained base model influence the performance of TAIL, and are certain pretrained weights more conducive to this kind of adaptation?* We initiated our investigation by analyzing the success rates of 40 Kitchen tasks using different pretrained weights for the spatial encoder. Apart from the CLIP-ViT pretrained encodings as we adopted in our main results, two other initialization of weights were considered: one sourced from the Visual Cortex 1 (VC-1) [227], recognized for being a leading pretrained model for embodied agent tasks, and another using randomly initialized weights. The language instruction encoder consistently utilized the CLIP text model. From the results in Fig. D.3, the VC-1 pretrained weights delivered performance on par with the CLIP-ViT encodings. Both considerably outperformed the randomly initialized weights, suggesting that large-scale

pretraining can indeed enhance downstream fine-tuning. We then study how does the pretrained base model influence the performance of TAIL.

D.3.3 Further Evaluations on TAIL with Different Base Models

To understand the influence of the base model's features on the performance of TAIL, we conducted additional evaluations. In Table D.3, the methods column showcases different configurations:

- **LoRA (CLIP):** The main setup we adopted in the experiment section 6.5, which keeps the pretrained CLIP encodings frozen across all the adaptation stages.
- **LoRA (CLIP with FFT):** Starting with the CLIP model, we applied FFT pretraining on the Kitchen task before using LoRA for subsequent adaptations. This helps us test out whether adaptation plasticity suffers after full fine-tuning as the only difference between this and the above method is the addition of full fine-tuning before using LoRA.
- **LoRA (VC-1 with FFT):** The VC-1 model, after FFT pretraining on the Kitchen task, was adapted using LoRA.
- **LoRA (Random with FFT):** A model with randomly initialized weights underwent FFT pretraining on the Kitchen task, followed by adaptation with LoRA.

All the pretrained encodings implemented in the same model architecture as described in Appendix Section D.1.

Observations from Table D.3 highlight several findings:

- **Dominance of Original CLIP:** The pure CLIP base model, when combined with LoRA, yielded the highest success rates across all task suites, suggesting the inherent quality and robustness of the original CLIP features for these tasks.

- **FFT's Mixed Impact:** While FFT pretraining aids in task-specific fine-tuning, when combined with CLIP, it leads to a degradation in performance. This could be attributed to FFT potentially diluting the comprehensive and rich features within CLIP while also reducing adaptation plasticity [173], especially when exposed to a more constrained domain with limited data.

- **VC-1's Comparable Performance:** The VC-1 model, though renowned in the domain of embodied agent tasks, delivered results that were only marginally better than the randomly initialized weights when both were subjected to FFT pretraining and then adapted with LoRA. This emphasizes the unique advantages of the original CLIP features.

Interestingly, it is observed that CLIP is pretrained on the most comprehensive dataset, followed by VC-1. In contrast, the model with random weights only underwent pretraining on the 40 Kitchen tasks. The success rates mirror this order, underscoring the idea that the efficacy of TAIL is closely tied to a base model pretrained with rich features on extensive datasets. So in summary, the choice of base model significantly affects the performance of TAIL, with CLIP's original features showing remarkable compatibility and resilience across various task suites

Table D.3: Evaluation results of FWT for LoRA with different pretrained model weights. The higher, the better. We highlight the best method with highest FWT as **bold**.

Method	Spatial	Goal	Object	Living Room	Study Room	Average
LoRA (CLIP)	**0.76 ± 0.02**	**0.79 ± 0.02**	**0.73 ± 0.14**	**0.73 ± 0.07**	**0.55 ± 0.11**	**0.71 ± 0.07**
LoRA (CLIP with FFT)	0.62 ± 0.04	0.67 ± 0.13	0.38 ± 0.08	0.32 ± 0.08	0.32 ± 0.01	0.46 ± 0.07
LoRA (Random with FFT)	0.38 ± 0.19	0.60 ± 0.06	0.37 ± 0.03	0.23 ± 0.01	0.47 ± nan	0.41 ± 0.07
LoRA (VC-1 with FFT)	0.56 ± 0.07	0.66 ± 0.08	0.25 ± 0.00	0.20 ± 0.06	0.48 ± 0.07	0.43 ± 0.05

D.3.4 Rank Size Ablation Study

In order to understand the impact of rank-size on adaptation performance, we conducted experiments using varying rank sizes for the LoRA and Bottleneck Adapter methods. The results, illustrated in Fig. D.4,

present the average success rates across the Spatial, Goal, and Object task suites. It is evident that increasing the rank size generally enhances performance up to a certain point. Beyond this optimal threshold, further increasing the rank size does not necessarily lead to higher success rates, potentially because of overfitting. Notably, in our continual learning context, the parallel insertion approach of LoRA consistently surpasses the sequential style of the Bottleneck Adapter method.

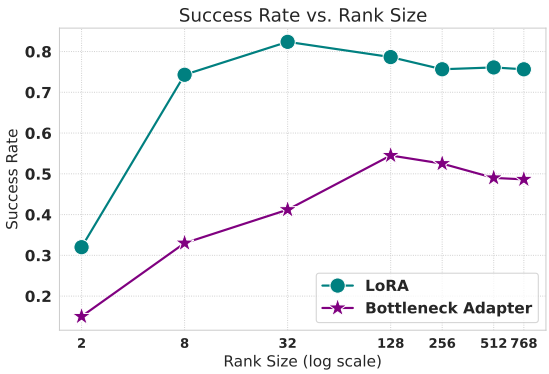


Figure D.4: Ablation study of the rank-size of LoRA and Bottleneck adapters. Increasing the rank size generally enhances performance up to a certain point. Beyond this optimal threshold, further increasing the rank size does not necessarily lead to higher success rates. The parallel insertion approach of LoRA consistently surpasses the sequential style of the Bottleneck Adapter method

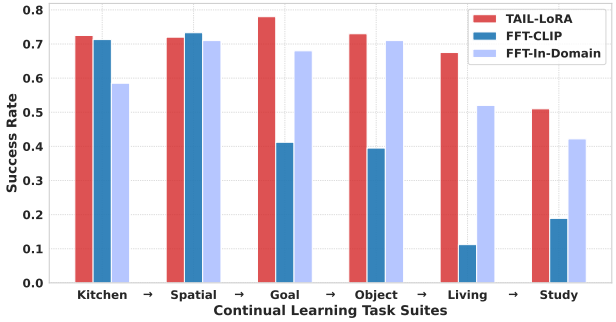


Figure D.5: Comparison for TAIL-LoRA, sequential FFT with pre-trained CLIP weights, and FFT-In-Domain. FFT-In-Domain is trained from scratch with task-suite demonstration data only, which saves a copy of the entire model for each task. FFT with CLIP excels in initial Kitchen and Spatial suites, highlighting the value of pretrained models; however, its performance declines in subsequent tasks, suggesting reduced adaptability. In contrast, TAIL-LoRA demonstrates consistent superior performance across all suites.

Additionally, we would like to note that our TAIL framework exhibits data adaptivity, suggesting that the rank size could be adjusted based on the quantity of adaptation data. In scenarios with smaller datasets, a smaller rank size could be more effective, and vice versa.

D.3.5 Comparison between Training from Scratch and Using Pretrained Models

Fig. D.5 compares the success rates across task suites for TAIL-LoRA, sequential FFT with pre-trained CLIP weights, and FFT-In-Domain. Unlike FFT-CLIP, FFT-In-Domain is trained from scratch with task-suite demonstration data only, i.e., we need to maintain a copy of the entire model for each task suite. There are three observations:

1. Pretrained Weights Advantage: In the initial Kitchen and Spatial task suites, FFT with CLIP pre-trained weights demonstrates a higher success rate compared to FFT trained from scratch. This indicates the effectiveness of leveraging pretrained models, particularly in the context of the Kitchen suite where the benefit is more pronounced.

2. Decline in Model Adaptability: Despite the initial advantage, sequential FFT with CLIP shows a marked decline in performance in the remaining four task suites - Goal, Object, Living, and Study. This trend may be indicative of a loss in model plasticity, where the pre-trained model performs well in the early stages but struggles to adapt to new tasks after the pre-trained weights are contaminated.

3. TAIL-LoRA's Consistent Performance: Throughout all the task suites, TAIL-LoRA with pre-trained CLIP consistently outperforms the other methods. This suggests that the LoRA approach, combined with the advantages of pretrained CLIP weights, provides a robust and adaptable framework capable of handling a variety of tasks with greater efficiency.

D.3.6 Ablation study for different integration style combinations

It's noteworthy that our method allows for the simultaneous use of multiple integration techniques [235]. This flexibility lets us explore the performance impact of combining LoRA (parallel integration), bottleneck adapter (sequential integration), and prefix token (concatenation). To this end, we conduct an ablation study for each of the combinations over the Spatial, Goal, and Object task suites. The experiment result is shown in Fig. D.6, where the y-axis is the averaged success rate.

A key finding is the critical role of LoRA (parallel integration) in enhancing adaptation performance. Combinations involving LoRA consistently outperform those without it. For instance, the standalone use of LoRA yields a comparable success rate w.r.t the combination with others. This pattern underscores LoRA's effectiveness, either used alone or in conjunction with other methods. In contrast, the combination

of Prefix and Adapter without LoRA results in a notably lower success rate (0.6641), highlighting LoRA's indispensability.

The integration of all three methods—Prefix, Adapter, and LoRA—achieves a success rate that is comparable to LoRA's standalone performance. This outcome suggests that while the combination of different integration methods does not detract from performance, LoRA remains the primary driver of successful adaptation. These findings emphasize the importance of LoRA in adapter weight integration strategies and provide valuable guidance for future approaches in this domain.

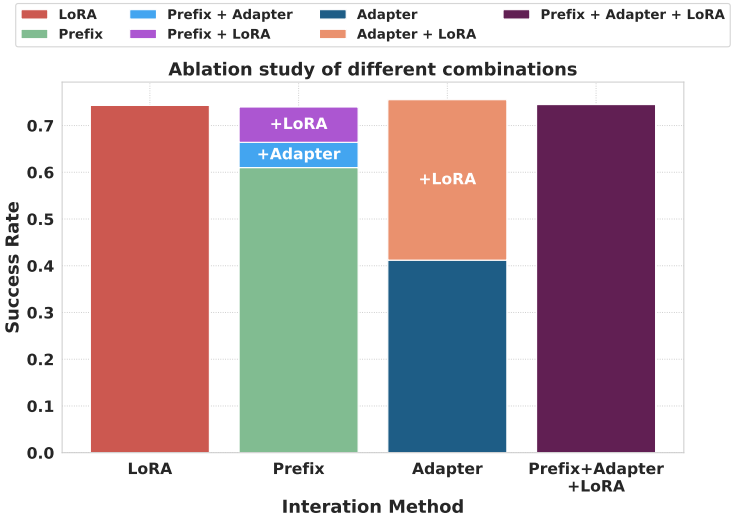


Figure D.6: Ablation study for integration style combinations. LoRA (parallel integration) plays a crucial role in enhancing adaptation performance, consistently outperforming methods without it. Whether used alone or in combination with other methods like Prefix and Adapter, LoRA shows superior effectiveness.

D.3.7 Detailed per-task results in the LIBERO-Long task suite

D.4 Evaluation Task Details

We list all the language instructions describing the tasks we adopted in our experiments below. Note that while certain tasks may share similar descriptions, they are not the same due to variations in the environment configurations (e.g., different spatial layouts, objects, or goal positions).

Table D.4: Adaptation results on 10 long horizon tasks. The ↑ symbol means the higher, the better. The BWT ↑ for TAIL methods are all 0 (no catastrophic forgetting). We highlight the best method (highest FWT ↑) in **bold**. FPF results were omitted due to its near-zero performance.

Task	Conventional Fine-Tuning Methods						TAIL-based Methods (**Ours**)			
	Full Fine-Tuning		Experience Replay		EWC		LoRA	Prefix	Bottleneck	RoboAdapter
	FWT ↑	BWT ↑	FWT ↑	BWT ↑	FWT ↑	BWT ↑	FWT ↑	FWT ↑	FWT ↑	FWT ↑
Task 1	0.42 ± 0.07	-	0.25 ± 0.12	-	0.38 ± 0.12	-	**0.62 ± 0.00**	0.38 ± 0.12	0.21 ± 0.14	0.12 ± 0.00
Task 2	0.58 ± 0.07	-0.42 ± 0.06	0.58 ± 0.07	-0.25 ± 0.10	0.54 ± 0.07	-0.38 ± 0.10	**0.75 ± 0.00**	0.58 ± 0.19	**0.75 ± 0.12**	0.50 ± 0.12
Task 3	0.71 ± 0.07	-0.50 ± 0.10	0.67 ± 0.07	-0.42 ± 0.19	0.38 ± 0.12	-0.46 ± 0.12	**0.96 ± 0.07**	0.88 ± 0.22	0.71 ± 0.19	0.50 ± 0.25
Task 4	**0.96 ± 0.07**	-0.57 ± 0.13	0.92 ± 0.07	-0.50 ± 0.20	0.75 ± 0.25	-0.43 ± 0.12	0.88 ± 0.00	0.71 ± 0.07	0.71 ± 0.19	0.58 ± 0.14
Task 5	0.21 ± 0.07	-0.67 ± 0.21	0.33 ± 0.14	-0.60 ± 0.25	0.17 ± 0.19	-0.50 ± 0.18	**0.62 ± 0.12**	0.17 ± 0.07	0.25 ± 0.00	0.29 ± 0.07
Task 6	**0.83 ± 0.19**	-0.57 ± 0.26	0.71 ± 0.19	-0.55 ± 0.25	0.50 ± 0.43	-0.42 ± 0.19	0.75 ± 0.12	0.79 ± 0.14	0.75 ± 0.00	0.75 ± 0.25
Task 7	0.17 ± 0.07	-0.62 ± 0.27	0.12 ± 0.00	-0.58 ± 0.25	0.04 ± 0.07	-0.44 ± 0.24	**0.54 ± 0.26**	0.38 ± 0.12	0.31 ± 0.09	0.33 ± 0.07
Task 8	0.42 ± 0.07	-0.55 ± 0.29	0.29 ± 0.07	-0.51 ± 0.28	0.12 ± 0.18	-0.46 ± 0.28	**0.75 ± 0.25**	0.67 ± 0.19	0.25 ± 0.18	0.50 ± 0.22
Task 9	0.17 ± 0.07	-0.54 ± 0.28	0.12 ± 0.05	-0.50 ± 0.28	0.00 ± 0.00	-0.41 ± 0.29	**0.38 ± 0.12**	0.08 ± 0.07	0.19 ± 0.09	0.21 ± 0.07
Task 10	0.33 ± 0.19	-0.50 ± 0.29	0.50 ± 0.02	-0.46 ± 0.29	0.12 ± 0.18	-0.38 ± 0.31	**0.79 ± 0.07**	0.50 ± 0.33	0.44 ± 0.09	0.42 ± 0.07
Average	0.48 ± 0.10	-0.55 ± 0.21	0.45 ± 0.09	-0.49 ± 0.23	0.30 ± 0.16	-0.43 ± 0.20	**0.70 ± 0.10**	0.51 ± 0.15	0.46 ± 0.11	0.42 ± 0.13

Task Suite	Instructions
Kitchen	close the top drawer of the cabinet close the top drawer of the cabinet and put the black bowl on top of it put the black bowl in the top drawer of the cabinet put the butter at the back in the top drawer of the cabinet and close it put the butter at the front in the top drawer of the cabinet and close it put the chocolate pudding in the top drawer of the cabinet and close it open the bottom drawer of the cabinet open the top drawer of the cabinet open the top drawer of the cabinet and put the bowl in it put the black bowl on the plate put the black bowl on top of the cabinet open the top drawer of the cabinet put the black bowl at the back on the plate put the black bowl at the front on the plate put the middle black bowl on the plate put the middle black bowl on top of the cabinet stack the black bowl at the front on the black bowl in the middle stack the middle black bowl on the back black bowl put the frying pan on the stove put the moka pot on the stove turn on the stove turn on the stove and put the frying pan on it close the bottom drawer of the cabinet close the bottom drawer of the cabinet and open the top drawer put the black bowl in the bottom drawer of the cabinet put the black bowl on top of the cabinet put the wine bottle in the bottom drawer of the cabinet put the wine bottle on the wine rack close the top drawer of the cabinet put the black bowl in the top drawer of the cabinet put the black bowl on the plate put the black bowl on top of the cabinet put the ketchup in the top drawer of the cabinet close the microwave put the yellow and white mug to the front of the white mug open the microwave put the white bowl on the plate put the white bowl to the right of the plate put the right moka pot on the stove turn off the stove

Table D.5: 40 Kitchen scene pretraining tasks

Task Suite	Instructions
Long-horizon (LIBERO 10)	put both the alphabet soup and the tomato sauce in the basket put both the cream cheese box and the butter in the basket turn on the stove and put the moka pot on it put the black bowl in the bottom drawer of the cabinet and close it put the white mug on the left plate and put the yellow and white mug on the right plate pick up the book and place it in the back compartment of the caddy put the white mug on the plate and put the chocolate pudding to the right of the plate put both the alphabet soup and the cream cheese box in the basket put both moka pots on the stove put the yellow and white mug in the microwave and close it
Spatial	pick up the black bowl between the plate and the ramekin and place it on the plate pick up the black bowl next to the ramekin and place it on the plate pick up the black bowl from table center and place it on the plate pick up the black bowl on the cookie box and place it on the plate pick up the black bowl in the top drawer of the wooden cabinet and place it on the plate pick up the black bowl on the ramekin and place it on the plate pick up the black bowl next to the cookie box and place it on the plate pick up the black bowl on the stove and place it on the plate
Goal	open the middle drawer of the cabinet put the bowl on the stove put the wine bottle on top of the cabinet open the top drawer and put the bowl inside put the bowl on top of the cabinet push the plate to the front of the stove put the cream cheese in the bowl turn on the stove
Object	pick up the alphabet soup and place it in the basket pick up the cream cheese and place it in the basket pick up the salad dressing and place it in the basket pick up the bbq sauce and place it in the basket pick up the ketchup and place it in the basket pick up the tomato sauce and place it in the basket pick up the butter and place it in the basket pick up the milk and place it in the basket
Living Room	pick up the alphabet soup and put it in the basket pick up the butter and put it in the basket pick up the milk and put it in the basket pick up the orange juice and put it in the basket pick up the tomato sauce and put it in the basket pick up the alphabet soup and put it in the tray pick up the butter and put it in the tray pick up the cream cheese and put it in the tray
Study Room	pick up the book and place it in the right compartment of the caddy pick up the book and place it in the front compartment of the caddy pick up the book and place it in the left compartment of the caddy pick up the book and place it in the right compartment of the caddy pick up the red mug and place it to the right of the caddy pick up the white mug and place it to the right of the caddy pick up the book in the middle and place it on the cabinet shelf pick up the book on the left and place it on top of the shelf

Appendix E

HAND

E.1 Environment Details and Hyperparameters

(a) CALVIN [237]

(b) Real-World WidowX-250

Figure E.1: We retrieve data from a prior dataset to train on *new scenes* in CALVIN. On our real-world WidowX-250 robot, we demonstrate real-world learning from HAND-retrieved trajectories along with real-time adaptation to long-horizon tasks.

E.1.1 CALVIN.

The CALVIN benchmark is built on top of the PyBullet [68] simulator and involves a 7-DOF Franka Emika Panda Robot arm that manipulates the scene. CALVIN consists of 34 tasks and 4 different environments (`ABCD`). All environments are equipped with a desk, a sliding door, a drawer, a button that turns on/off an LED, a switch that controls a lightbulb and three different colored blocks (red, blue and pink). These environments differ from each other in the texture of the desk and positions of the objects. CALVIN provides 24 hours of tele-operated unstructured play data, 35% of which are annotated with language

descriptions. We utilize this 35% as a natural way to obtain a smaller subset of the data as the full dataset is very large, but we do not use the task-oriented language instructions. In total, $\mathcal{D}_{\text{play}}$ corresponds to $\sim$ 17k trajectories for our experiments.

We evaluate on the following tasks:

- **Close Drawer.** For this task, the arm is required to push an opened drawer and close it. The drawer's degree of openness is randomized.
- **Move Slider Left.** This task requires the robot arm to move a slider located on the desk from the right to the left. The slider position is randomized.
- **Turn On Led.** In this task, the robot arm needs to navigate its way to a button and press down on it such that an LED turns on.
- **Lift Blue Block Table.** For this task, the robot arm needs to pick up a blue block from the table. The location of the blue block on the table is randomized.

E.1.2 Real Robot Experimental Setup

Figure E.2: We evaluate HAND on 5 different real robot tasks. The last two are long-horizon tasks, requiring more than 100 timesteps of execution.

Hardware Setup. We evaluate HAND on a real-world multi-task kitchen environment using the WidowX robot arm. The WidowX is a 7-DoF robot arm with a two-fingered parallel jaw gripper. Our robot environment setup is shown in Figure E.1. We use an Intel Realsense D435 RGBD camera as a static external camera and a Logitech webcam as an over-the-shoulder camera view. We use a Meta Quest 2 VR headset for teleoperating the robot.

Task-agnostic play dataset. Our play dataset contains a total of 50k transitions collected at 5hz. To encourage diverse behaviors and motions, human teleoperators were instructed to freely interact with the available objects in the scene without being bound to specific task goals.

Evaluation protocol. The agent is allocated a 100 timestep budget to complete each task. Furthermore, we introduce distractor objects in the scene that are not part of the task so that the policy does not just memorize the expert demonstrations. Moreover, movable task object positions are randomized in a fixed region if applicable. We evaluate on four manipulation tasks described below:

- **Reach Block.** In this task, the robot arm must reach and hover directly above a green block placed on the table. Success is achieved when the gripper remains positioned clearly above the block. Partial success is awarded if the gripper end-effector touches the block without hovering steadily above it.
- **Push Button.** This task requires the robot arm to press the right-side button on a stovetop. Success is achieved upon pressing the button. Partial success is awarded if the robot arm approaches sufficiently close to the button without making contact.

- **Close Microwave.** This task requires the robot to close a microwave door from various starting angles. Partial successe is awarded if the robot pushes the door without completely closing it. A successful closure is confirmed by an audible click sound.
- **Put K-Cup in Coffee Machine.*** In this task, the robot needs to first pick up the Keurig cup and then transport it to the coffee machine and insert the cup into the cup holder. This task requires precision low-level control as the Keurig cup is small, making it difficult to grasp reliably. Additionally, the cup holder on the coffee machine is just large enough to fit the Keurig cup, leaving small margin of error during the insertion. The coffee machine is fixed to the kitchen stovetop, while the initial location of the Keurig cup is randomized. Given the difficulty of the task, we provide partial success for successfully grasping the Keurig cup.
- **Blend Carrot.** The robot first picks up a toy carrot and then drops it into the blender. Once the carrot is inside the blender, it will press a button at the blender base to activate the blender and hold the button for 2 seconds. The location of the blender is static, but the carrot is randomized. Partial success is provided for picking up the carrot and also successfully dropping it into the blender.

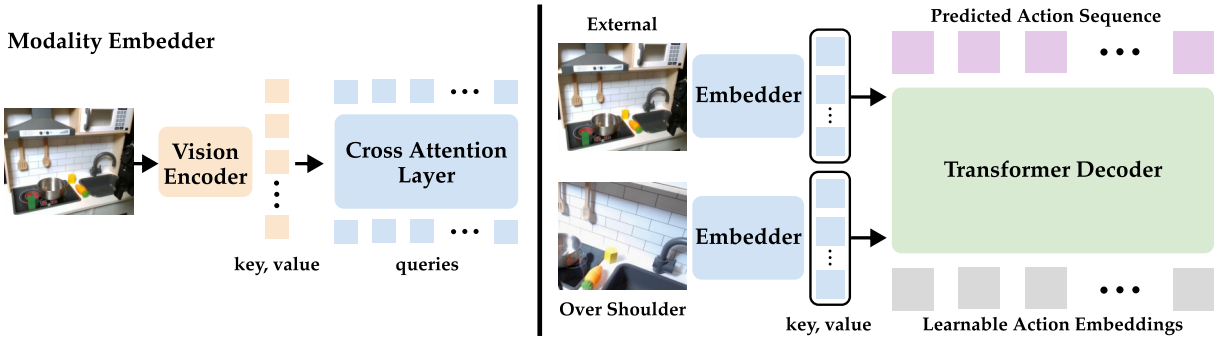


Figure E.3: (Left) Learnable image embeddings following [365]. (Right) The learned image embeddings for each modality are concatenated and provided to a transformer decoder similar to [410]. We also perform action chunking with a chunk size of 5 timesteps for 1 second of execution.

*https://www.samsclub.com/p/members-mark-gourmet-kitchen-appliances-playset/P990340349?xid=plp_product_2

Robot Policy. For our policy, we are inspired by the architectural components introduced in Wang et al. [365] and Zhao et al. [410]. A diagram of our policy architecture is shown in Figure E.3. For both external and over-the-shoulder RGB images, we use a pretrained ResNet to first extract $\times 7 \times 7$ feature maps and flatten these features across the spatial dimension to create a sequence of d_v dimension tokens where d_v is the output dimension of ResNet. In particular, we use ResNet18 where $d_v = 512$. We feed as input to a causal transformer decoder a sequence learnable action tokens with dimension d. We use the flattened image feature map as the keys and values and apply a cross-attention between the image features and learnable tokens. We concatenate all modality tokens and add additional modality-specific embeddings and sinusoidal positional embeddings.

The policy base is a transformer decoder similar to the one used in ACT [410]. The input sequence to the transformer is a fixed position embedding, with dimensions $k \times 512$ where k is the chunk size and the keys and values are the combined image tokens from the stem. Given the current observation, we predict a chunk of 5 actions, which corresponds to 1 second of execution. During inference time, we also apply temporal ensembling similar to [410] with $m = 0.5$, which controls the weight of previous actions.

We train the policy for 20k update steps with batch size of 256 and a learning rate of $3e^{-4}$ (around 2 hours of wall time). For behavior cloning policies, the action dimension is 7 comprising of the robot joint pose and the gripper state.

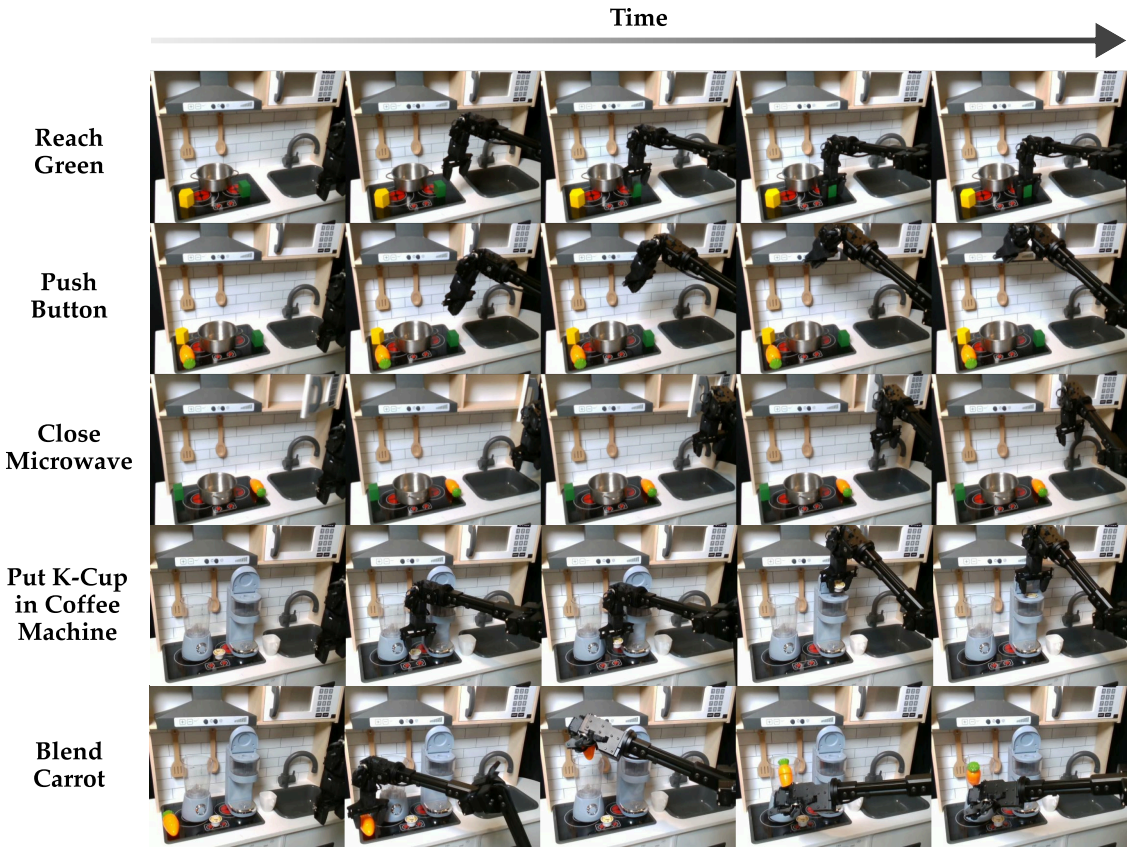


Figure E.4: Task Rollouts

E.2 HAND Algorithm

Algorithm 7 HAND Full Algorithm

Require: Hand demonstrations $\mathcal{D}_{\text{hand}}$, offline play datsaet $\mathcal{D}_{\text{play}}$, CoTracker3, Molmo-7B, # retrieved sub-trajectories K, threshold ϵ, DINO, # visual filtered sub-trajectoies M

/ Policy Pretraining */*

Train π_{base} on $\mathcal{D}_{\text{play}}$ using regular behavior cloning loss $\mathcal{L}_{BC}$

/ Sub-Trajectory Pre-processing */*

$\mathcal{T}_{\text{hand}} \leftarrow \texttt{SubTrajSegmentation}(\mathcal{D}_{\text{hand}}, \epsilon)$ ▷ Heuristic demo segmentation

$\mathcal{T}_{\text{play}} \leftarrow \texttt{SubTrajSegmentation}(\mathcal{D}_{\text{play}}, \epsilon)$ ▷ Heuristic demo segmentation

/ Retrieval using S-DTW and 2D Hand Paths Section 7.3.2 */*

$\mathcal{D}_{\text{retrieved}} \leftarrow \{\}$

for $\tau_{\text{hand}} \in \mathcal{T}_{\text{play}}$ **do**

  $o^{\text{hand}}_{1:H} \leftarrow$ image obs sequence of τ_{hand}

  for $\tau_{\text{play}} \in \mathcal{T}_{\text{play}}$ **do**

    $o^{\text{play}}_{1:T} \leftarrow$ image obs sequence of τ_{play}

    / Visual Filtering */*

    Compute $\text{C}_{\text{visual}}(o^{\text{hand}}_{1:H}, o^{\text{play}}_{1:T})$ with DINO ▷ Equation (7.1)

  $\mathcal{T}^{M}_{\text{play}} \leftarrow M$ sub-trajectories with lowest C_{visual}

  for $\tau_{\text{play}} \in \mathcal{T}^{M}_{\text{play}}$ **do**

    $o^{\text{play}}_{1:T} \leftarrow$ image obs sequence of τ_{play}

    $(x, y)_{\text{hand}} = \text{Molmo}(o_{H/2}), (x, y)_{\text{play}} = \text{Molmo}(o_{T/2})$ ▷ Get middle frame query point

    $p_{\text{hand}} = \{(x_t, y_t)_{\text{hand}}\}_1^H = \text{CoTracker3}((x, y)_{\text{hand}})$ ▷ Track hand point

    $p_{\text{play}} = \{(x_t, y_t)_{play}\}_1^T = \text{CoTracker3}((x, y)_{\text{play}})$ ▷ Track robot gripper point

    $p_{\text{hand}} = p_{\text{hand}}[:-1] - p_{\text{hand}}[1:]$ ▷ Convert p_{hand} and p_{play} to *relative 2D paths*

    $p_{\text{play}} = p_{\text{play}}[:-1] - p_{\text{play}}[1:]$

    $(\text{C}_{\text{path}}, \tau^{\text{play}}_{i:j}) \leftarrow \text{S-DTW}(p_{\text{hand}}, p_{\text{play}})$ ▷ Path cost and corresponding retrieved sequence

  Add K lowest C_{path} $\tau^{\text{play}}_{i:j}$ sub-trajectories to $\mathcal{D}_{\text{retrieved}}$

/ Parameter-Efficient Policy Fine-tuning */*

Insert *task-specific* adapter LoRA layers θ in π_{base}

Update π_{base} on $\mathcal{D}_{\text{retrieved}}$ with loss $\mathcal{L}_{BC;\theta}$ ▷ Equation (7.2)

return π_θ

E.3 User Studies

E.3.1 Efficiency of Hand Demonstrations

Teleoperation **Hand Demo**

Figure E.5: **Efficient Demonstrations.** Two users, unfamiliar with HAND are asked to collect trajectories either via teleoperation (Left) or using their hands (Right). HAND retrieval achieves a 50% success rate with the same amount of demonstrations using $3\times$ less time. STRAP retrieval is unable to reach 50% even when provided with more expert demonstrations.

In our first study, two users collect 10 demonstrations each either by manually teleoperating using a VR controller or by providing a hand demonstration. For manual teleoperation, we explain to the users how to operate the robot using the VR controller and allow them a couple trials to get accustomed to the interface. For hand demonstrations, we ask the users to mimic the trajectory of the robot end effector using their hands. Figure E.5 shows an example of a user performing both forms of demonstrations. We observe that providing hand demonstrations is significantly more time efficient (over $3\times$) compared to manual teleoperation. Furthermore, with just a single hand demonstration, we are able to learn a performant policy with 50% success rate, while STRAP struggles even when provided 5 expert demonstrations.

E.3.2 Fast Adaptation to Long-Horizon Tasks

We conduct a small study demonstrating that HAND enables real-time fast, adaptation to unseen downstream tasks. Snapshots at various stages of this experiment is shown in Figure E.6. In our study, we

Figure E.6: **Fast Adaptation.** We conduct a small-scale user study to demonstrate HAND's ability to learn robot policies in real-time. From providing the hand demonstration (Left), to retrieval and fine-tuning a base policy (Middle), to evaluating the policy (Right), we show that HAND can learn to solve the `Blend Carrot` task with over 70% success rate in less than 3 minutes.

measure the total time required for a user to provide a hand demonstration of a new target task to evaluating the performance of a fine-tuned policy. The hand demonstration is simple to provide and typically takes between $10 - 15$ seconds to collect. Data preprocessing, which involves computing the 2D path features of the hand demonstration and performing retrieval, takes around $30 - 40$ seconds. We assume that the offline play dataset is already preprocessed prior to the study and we do not include this time in our estimate. We also assume a base policy has already been trained on this data; however, it performs poorly on the target task. We fine-tune the base policy with 4 LoRA adapter layers for 1000 batch updates, which takes $\sim$ 2 minutes on a NVIDIA 4070 GPU. The resulting policy, which took less than 3 minutes to train and achieves over 70% success rate, highlighting the efficacy of HAND for real-time policy learning. An uncut video of this study can be found on our project website at https://handretrieval.github.io/.

E.4 Qualitative Retrieval Analysis

In Figure E.7, we provide more qualitative results comparing STRAP retrieval results to HAND on each of our real robot tasks. Across all tasks, HAND retrieves more relevant trajectories that perform the task demonstrated by the human hand.

E.5 CALVIN Results

Method	K=25	K=50	K=100	K=250
With Expert				
FT	0.425 ± 0.059	-	-	-
Flow	0.694 ± 0.089	0.797 ± 0.045	0.633 ± 0.127	0.747 ± 0.039
STRAP	0.481 ± 0.119	0.286 ± 0.073	0.703 ± 0.075	0.600 ± 0.085
Without Expert				
π_{base}	0.233 ± 0.024	-	-	-
CLIP	0.003 ± 0.004	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Flow	0.808 ± 0.080	0.831 ± 0.058	0.533 ± 0.106	0.653 ± 0.055
STRAP	0.000 ± 0.000	0.011 ± 0.010	0.006 ± 0.008	0.031 ± 0.004
HAND(+3D,-VF,-CW)	0.994 ± 0.004	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
HAND(-VF,-CW)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
HAND(-VF)	1.000 ± 0.000	1.000 ± 0.000	0.997 ± 0.004	1.000 ± 0.000
HAND	0.828 ± 0.169	0.464 ± 0.061	0.536 ± 0.082	0.436 ± 0.136

Table E.1: Performance with and without expert demonstrations

Method	K=25	K=50	K=100	K=250
With Expert				
FT	0.564 ± 0.309	-	-	-
Flow	0.092 ± 0.038	0.086 ± 0.017	0.156 ± 0.046	0.039 ± 0.039
STRAP	0.053 ± 0.034	0.075 ± 0.012	0.111 ± 0.014	0.094 ± 0.037
Without Expert				
π_{base}	0.011 ± 0.010	-	-	-
CLIP	0.017 ± 0.024	0.033 ± 0.047	0.006 ± 0.004	0.031 ± 0.024
Flow	0.000 ± 0.000	0.247 ± 0.116	0.094 ± 0.046	0.053 ± 0.014
STRAP	0.058 ± 0.018	0.122 ± 0.022	0.075 ± 0.025	0.028 ± 0.024
HAND(+3D,-VF,-CW)	0.028 ± 0.008	0.047 ± 0.010	0.192 ± 0.049	0.139 ± 0.040
HAND(-VF,-CW)	0.186 ± 0.088	0.081 ± 0.017	0.364 ± 0.149	0.619 ± 0.092
HAND(-VF)	0.069 ± 0.042	0.167 ± 0.056	0.200 ± 0.123	0.325 ± 0.014
HAND	0.647 ± 0.229	0.483 ± 0.041	0.636 ± 0.103	0.431 ± 0.107

Table E.2: Performance with and without expert demonstrations

Method	K=25	K=50	K=100	K=250
With Expert				
FT	0.000 ± 0.000	-	-	-
Flow	0.131 ± 0.085	0.344 ± 0.092	0.697 ± 0.082	0.581 ± 0.134
STRAP	0.200 ± 0.147	0.125 ± 0.042	0.056 ± 0.017	0.372 ± 0.220
Without Expert				
π_{base}	0.036 ± 0.014	-	-	-
CLIP	0.025 ± 0.035	0.006 ± 0.008	0.019 ± 0.016	0.000 ± 0.000
Flow	0.017 ± 0.024	0.011 ± 0.008	0.364 ± 0.147	0.436 ± 0.031
STRAP	0.500 ± 0.131	0.600 ± 0.184	0.525 ± 0.150	0.633 ± 0.112
HAND(+3D,-VF,-CW)	0.333 ± 0.111	0.661 ± 0.093	0.814 ± 0.059	0.489 ± 0.136
HAND(-VF,-CW)	0.675 ± 0.065	0.719 ± 0.155	0.886 ± 0.032	0.431 ± 0.103
HAND(-VF)	0.428 ± 0.016	0.467 ± 0.138	0.828 ± 0.058	0.881 ± 0.034
HAND	0.136 ± 0.102	0.278 ± 0.073	0.186 ± 0.051	0.094 ± 0.017

Table E.3: Performance with and without expert demonstrations

E.6 Real Robot Results

Task \ Method	π_{base}	STRAP			HAND		
		$K{=}10$	$K{=}25$	$K{=}50$	$K{=}10$	$K{=}25$	$K{=}50$
Reach Green Block	1.0	2.5	2.0	2.5	6.0	7.5	5.0
Press Button	0.0	5.5	5.0	2.5	8.5	5.0	4.0
Close Microwave	0.5	5.0	2.5	4.0	7.0	8.0	4.5

Table E.4: Success rates out of 10 trials per task.

Task \ Method	π_{base}	STRAP			HAND		
		$K{=}10$	$K{=}25$	$K{=}50$	$K{=}10$	$K{=}25$	$K{=}50$
Reach Green Block	1.0	3.0	1.0	1.0	6.5	7.0	6.0
Press Button	0.0	1.5	0.0	0.5	4.5	6.0	3.5
Close Microwave	0.5	0.0	0.0	0.0	8.0	4.0	1.0

Table E.5: Success rates out of 10 trials per task.

Figure E.7: **Qualitative Retrieval Examples.** We show the top 5 matches from $\mathcal{D}_{\text{play}}$ for STRAP (top) and HAND (bottom) provided the hand demonstration for each of our evaluation tasks.

Appendix F

BOSS

F.1 Dataset and Environment Details

F.1.1 ALFRED

F.1.1.1 Dataset Details

We base our dataset and environment on the ALFRED benchmark [328]. ALFRED originally contains over 6000 full trajectories collected from an expert planner following a set of 7 high-level tasks with randomly sampled objects (e.g., *"pick an object and heat it"*). Each trajectory has three crowdsourced annotations, resulting in around 20k distinct language-annotated trajectories. We separate these into only the primitive skill trajectories, resulting in about 141k language-annotated trajectories. Following Zhang et al. [404], we merge navigation skills (e.g., *"Walk to the bed"*) with the skill immediately following them as these navigation skills make up about half of the dataset, are always performed before another skill, and are difficult to design online RL reward functions for that work across all house floor plans given only the information in the dataset for these skills. After this processing step, the resulting dataset contains 73k language-annotated primitive skill trajectories.

F.1.1.2 RL Environment Details

We modified ALFRED similarly to Zhang et al. [404] and Pashevich, Schmid, and Sun [273] to make it suitable for policy learning by modifying the action space to be fully discrete, with 12 discrete action choices and 82 discrete object types.

Furthermore, we rewrote reward functions for all primitive skill types ("CoolObject", "PickupObject", "PutObject", "HeatObject", "ToggleObject", "SliceObject", "CleanObject") so that rewards can be computed independently of a reference expert trajectory. While our rewards depend on the ground truth primitive skill type, no agents are allowed access to what the underlying true primitive skill type is. All of our reward function are sparse, with 1 for a transition that completes primitive skill and 0 for all other transitions.

F.1.1.3 Evaluation Tasks

We generate evaluation tasks by randomly sampling 10 tasks each for 4 unseen ALFRED floor plans, resulting in 40 total tasks unseen tasks requiring anywhere from 2-8 primitive skills to complete. The tasks for each floor plan are sampled randomly from the Valid-Unseen ALFRED dataset collected in these plans with the specific object arrangements, and we use the high-level task language descriptions collected by humans for ALFRED as our task descriptions for language-conditioned zero-shot evaluation. See Figure F.1 for a histogram of task lengths.

F.1.2 Language Model Prompts

We use two prompts when using the LLM for two different purposes. The main purpose of the LLM is to propose a distribution over next skills to chain with currently executed skills during skill bootstrapping (Section 8.3.2). Thus, we pass skills in the given skill library Z into the prompt and ask it to predict the next skill. We also include a fixed set of 7 in-context examples from a random sample of different tasks from the ALFRED training dataset. The prompt for bootstrapping is shown in Figure F.2.

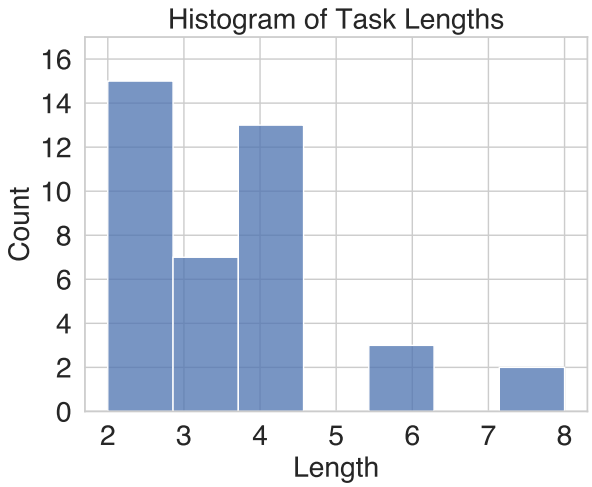


Figure F.1: Task lengths regarding the number of primitive skills needed to chain together to solve the task.

We also generate summaries (see Section 8.3.2 and appendix Appendix F.2.3) for *composite* skill annotations with the LLM. These summaries are used to label newly chained longer-horizon skills before adding them back to the skill library. We show the prompt for this in Figure F.3.

F.2 Training Implementation details and Hyperparameters

We implement IQL [171] as the base offline RL algorithm to pre-train on primitive skill data for all methods, baselines, and ablations, due to its strong offline and finetuning performance on a variety of dense and sparse reward environments.

The IQL policy is trained to maximize the following objective:

$$e^{\beta(Q(s,a)-V(s))} \log \pi(a|s),$$

which performs advantage-weighted regression [278] with an inverse temperature term β. Q and V are trained on (s, a, s', r, a') tuples from the dataset rather than sampling a policy for a' to mitigate issues

Examples of common household tasks and their descriptions:
Task Steps: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.
Task: Put the box with keys on the sofa.

Task Steps: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.
Task: Put a cooled slice of lettuce on the counter.

Task Steps: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.
Task: Put a book on the couch.

Task Steps: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.
Task: Put the cleaned fork in a drawer.

Task Steps: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.
Task: Put the box of tissues on the barred rack.

Task Steps: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.
Task: Put a heated glass on the wooden rack.

Task Steps: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.
Tasks: Look at the box under the lamp light.

Predict the next skill correctly by choosing from the following skills: [SKILL 1 IN LIBRARY], [SKILL 2 IN LIBRARY], ...
Task Steps: 1. [SKILL 1 EXECUTED SO FAR] 2. [SKILL 2 EXECUTED SO FAR] ... N. ____

Figure F.2: Prompt for the LLM for next skill proposal (Section 8.3.2). Text is generated after listing out all skills completed so far.

Instructions: give a high-level description for the following steps describing common household tasks.

Task Steps: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.
Summary: Put the box with keys on the sofa.

Task Steps: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.
Summary: Put a cooled slice of lettuce on the counter.

Task Steps: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.
Summary: Put a book on the couch.

Task Steps: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.
Summary: Put the cleaned fork in a drawer.

Task Steps: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.
Summary: Put the box of tissues on the barred rack.

Task Steps: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.
Summary: Put a heated glass on the wooden rack.

Task Steps: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.
Summary: Look at the box under the lamp light.

Task Steps: 1. [SKILL 1] 2. [SKILL 2] 3. [SKILL 3] ...
Summary:

Figure F.3: Prompt for the LLM to summarize completed skills into high-level *composite* annotations, following Zhang et al. [404].

with critic function overestimation common in offline RL. We detail shared training and implementation details below, with method-specific information and hyperparameters in the following subsections.

F.2.1 ALFRED Environment

We implement the same observation and action space as Zhang et al. [404]. Details are listed below.

Observation space. The observations given to agents are 300×300 RGB images. For all methods, we first preprocess these images by sending them through a frozen ResNet-18 encoder [133] pre-trained on ImageNet, resulting in a $512 \times 7 \times 7$ observation.

Action space. The agent chooses from 12 discrete low-level actions. There are 5 navigation actions: `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, and `LookDown` and 7 interaction actions: `Pickup`, `Put`, `Open`, `Close`, `ToggleOn`, `ToggleOff`, and `Slice`. For interaction actions the agent additionally selects one of 82 object types to interact with, as defined by Pashevich, Schmid, and Sun [273]. In total, the action space consists of $5 + 7 * 82 = 579$ discrete action choices. For all methods, due to the large discrete action space, we perform the same action masking as Zhang et al. [404] to prevent agents from taking actions that are not possible by using ground truth object properties given by the ALFRED simulator for each object in the scene. For example, we do not allow the agent to `Close` objects that aren't closeable or `ToggleOn` objects that can't be turned on.

Policy and critic networks. We use the transformer architecture (and hyperparameters) used by Episodic Transformers (ET) [273] for our policy and critic networks. We implement all critics (two Q functions and one V) with a shared backbone and separate output heads. Additionally, we use LayerNorms [18] in the MLP critic output heads as recommended by Ball et al. [23]. All networks condition on tokenized representations of input language annotations.

Hyperparameters. Hyperparameters were generally selected from tuning the Oracle baseline to work as best as possible, then carried over to all other methods. Shared hyperparameters for all methods (where

applicable) for pre-training on primitive skills are listed below. Any unlisted hyperparameters or implementation details are carried over from Pashevich, Schmid, and Sun [273]:

Param	Value
Batch Size	64
# Training Epochs	150
Learning Rate	1e-4
Optimizer	AdamW
Dropout Rate	0.1
Weight Decay	0.1
Discount γ	0.97
Q Update Polyak Averaging Coefficient	0.005
Policy and Q Update Period	1 per train iter
IQL Advantage Clipping	[0, 100]
IQL Advantage Inverse Temperature β	5
IQL Quantile τ	0.8
Maximum Observation Context Length	21

When fine-tuning policies (for Oracle, CIC, and BOSS), we keep hyperparameters the same. We fine-tune one policy per floor plan (zero-shot evaluating on 10 tasks in each floor plan) in our ALFRED task set so that the aggregated results are reported over 4 runs per seed. For methods that use a skill library (BOSS, Saycan, Saycan+P), all available primitive skills across all evaluation tasks in each floor plan compose the starting skill library, resulting in anywhere from 15-40 available skills depending on the floor plan.

Additionally, when finetuning the Oracle baseline along with BOSS and its ablations, we sample old data from the offline dataset and newly collected data at equal proportions in the batch, following suggestions from [23]. We do not do this for CIC when finetuning with its unsupervised RL objective because

the language embeddings from the old data are not compatible with the online collected data labeled with CIC-learned skill embeddings. Fine-tuning hyperparameters follow:

Param	Value
# Initial Rollouts	50
# Training Steps to Env Rollouts Ratio	15
ϵ in ϵ-greedy action sampling	0.05
Discrete action sampling	True
# Parallel Rollout Samplers	10

F.2.2 Real Robot Environment

The input observation from the environment includes environment RGB input and robot states. The RGB input consists of the third-person view RGB images from a Logitech Pro Webcam C920 cropped to 224×224×3, and wrist view images from an Intel RealSense D435. We use a pretrained R3M [252] model to get the latent representation for each view. The robot states include the robot's end-effector position, velocity, and gripper state. The end-effector position and velocity are two continuous vectors, and the gripper state is a one-hot vector, which presents OPEN, CLOSE, or NOT MOVE. We concatenate the RGB latent representations and robot states together as environment states.

The policy is language conditioned, and we use a pre-trained sentence encoder to encode the language annotation to a 384-dimensional latent vector. The pretrained sentence encoder we use is `all-MiniLM-L12-v2` from the `SentenceTransformers` package [298].

The total state input dimension is 2048 (third-person R3M) + 2048 (wrist R3M) + 15 (robot state input) + 384 (language latent representation) = 4495.

Action space. The action space of the robot encompasses the difference in the end effector position between each time step, along with discrete open and close signals for the gripper. These actions are transmitted to the robot with 10HZ and interpreted as desired joint poses using PyBullet's inverse kinematics module.

In line with [410], we adopt the Action Chunking method to train an autoregressive policy. Our policy utilizes an LSTM model to predict the next 15 actions, given the initial observation as input, denoted as $\pi(a_{t:t+15}|s_t)$. Both our Q and Value networks are recurrent as well, estimating rewards on a per-timestep basis for each action in the sequence. Similar to the policy, these networks only have access to the observation preceding the action sequence initiation.

Due to the gripper action space is discrete and imbalanced distributed in the dataset, we reweigh gripper loss inversely proportionally to the number of examples in each class.

F.2.3 Additional BOSS Implementation Details

Here we continue discussion of BOSS in detail. In the main text in Section 8.3.2 we mention that we add learned skills back to the agent's skill repertoire and then train on collected experience gathered from each rollout. Here, we detail exactly how we do that.

Labeling new composite skills. Finally, after we have finished attempting a composite skill chain, we need a natural language description for it so we can train the language-conditioned policy on this new composite skill. We ask the LLM to generate high-level task descriptions of the annotations of the two skills the agent has just attempted to chain together like proposed by Zhang et al. [404] for offline policy pre-training. Doing so will allow the agent to learn skills at a higher level of text abstraction, allowing the agent to operate on more natural evaluation task specifications. For example, humans are more likely to ask an agent to "Make coffee" than to say "Get a coffee pod. Put the coffee pod in the machine. Fill it up with water..."

We give the LLM a prompt similar to the one for generating next skills. For example, if our agent has just completed two skills: *"Pick up the spoon"*, *"Put the spoon on the counter"*, we ask the LLM to summarize "1. PICK UP THE SPOON. 2. PUT THE SPOON ON THE COUNTER.", and the LLM can generate *"put a spoon on the counter."* We denote the generated language annotation for this combined skill composed of the annotations of z^1 and z^2 as z'. We then add z' as a new composite skill to Z for the agent to possibly sample from again.

Training on new skill data. After the agent has finished a rollout in the environment, it trains on the experience gathered. There are three types of data that we add to the agent's replay buffer from its rollout data:

1. The trajectory of the attempted skill chain which is collected only if the entire first skill is successfully executed (regardless if it is a primitive skill or a chain of them) since only then will another skill be used for chaining. The label for this trajectory is produced by the LLM.

2. The trajectory of the composite skill but with a label generated by concatenating the primitive skill annotations as a sequence of sentences of their language annotations. This trajectory ensures that the agent receives a description for the collected composite trajectory that specifies the exact primitive skills that make it up, in order. This is useful because the LLM-generated high-level skill description may not describe certain steps. Those steps are explicitly spelled out in this new label.

3. Trajectories for all lowest-level primitive skills executed during the rollout. These correspond to the original set of skills the policy was equipped with and will help the policy continue to finetune its ability to execute its original primitive skills.

After the rollout, we add these trajectories to the agent's replay buffer.

Other details. When performing skill bootstrapping in the ALFRED environment, we set a max time limit (T in Algorithm 8) for 40 timesteps per primitive skill. For simplicity, we restrict M, the max number of skills to chain, to be 2 during skill bootstrapping rollouts. We also restrict the second skill to be chained to only the set of primitive skills so that the agent can only learn new skill chains that are one primitive skill longer than the first sampled skill. Note that this does not restrict the agent from sampling composite skills it has learned during bootstrapping as first skills upon initialization.

One final implementation detail is with respect to how we map LLM next skill proposals to existing skills in the skill library Z. We found that pre-trained sentence embedding models generally seem to put great emphasis on the *nouns* of skill annotation sentences in ALFRED, instead of the verb. Therefore, all sentence embeddings models we initially experimented with (up to the 11B parameter model FLAN-T5-XXL [62]) would have a tendency to map LLM generations such as *"Place the apple in the sink"* to skills with *different verbs* as long as the nouns were the same, such as *"Pick up the apple from the sink"*. These skills are clearly very different, so this presented a problem to us initially. To solve this, we settled on using an NLP library* to extract the main verb of sentences and then added that same verb as a prefix to each sentence before embedding with the sentence embedding model. For example, *"Place the apple in the sink"* $\rightarrow$ *"PLACE: Place the apple in the sink."* With this change, the aforementioned issue was addressed in most cases and we could use much smaller sentence embedding models (`all-mpnet-v2` from the SentenceTransformers package [298]).

Training Time and Hardware Requirements We perform experiments on a server with 2 AMD EPYC 7763 64-Core Processors, and 8 RTX 3090 GPUs. Pre-training the policies takes around 10 hours with just a single RTX 3090 and 4 CPU threads for parallel dataloading.

Skill bootstrapping experiments require just 1 GPU with sufficient VRAM to run inference with our LLM, along with 4 available CPU threads for parallel dataloading and environment rollouts. In practice, a

*https://github.com/chartbeat-labs/textacy

single RTX 3090 is sufficient for our experiments using LLaMA-13B with 8-bit inference [79] on ALFRED, requiring around 3-5 days of training, mainly due to the speed of the underlying simulator used in ALFRED.

F.2.4 CIC Implementation

For fairness in our experimental comparison, we implement CIC [178] by using its objective to train a policy pre-trained on the same dataset as BOSS; thus, the CIC agent is first initialized with a set of sensible behaviors. Since CIC operates on a fixed latent space, we modified the critic and policy architectures so that they operate on fixed-length, 768-dimensional embeddings of language inputs from the same sentence embedding model used for skill bootstrapping [298] instead of on variable length tokenized language representations.

CIC-specific hyperparameters follow:

Param	Value
CIC K-means K	12
CIC K-means avg	True
CIC Hidden Dim	1024
CIC Latent Skill Dim	768
CIC Temp	0.5
CIC Skill Projection Layer	True
# Timesteps for each skill rollout before reset	200

F.2.5 SayCan Implementation

We implement SayCan [7] by combining the prompt from SayCan with ours. We use the same in-context examples except but convert them to a human-robot conversation. All other details are the same, including the LLM that we use in this comparison (LLaMa-13b [352]). The Saycan prompt follows below:

Robot: Hi there, I'm a robot operating in a house. Robot: You can ask me to do various tasks and I'll tell you the sequence of actions I would do to accomplish your task.

Human: How would you put the box with keys on the sofa?

Robot: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.

Human: How would you put a cooled slice of lettuce on the counter?

Robot: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.

Human: How would you put a book on the couch?

Robot: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.

Human: How would you put the cleaned fork in a drawer?

Robot: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.

Human: How would you put the box of tissues on the barred rack?

Robot: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.

Human: How would you put a heated glass on the wooden rack?

Robot: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.

Human: How would you look at the box under the lamp light?

Robot: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.

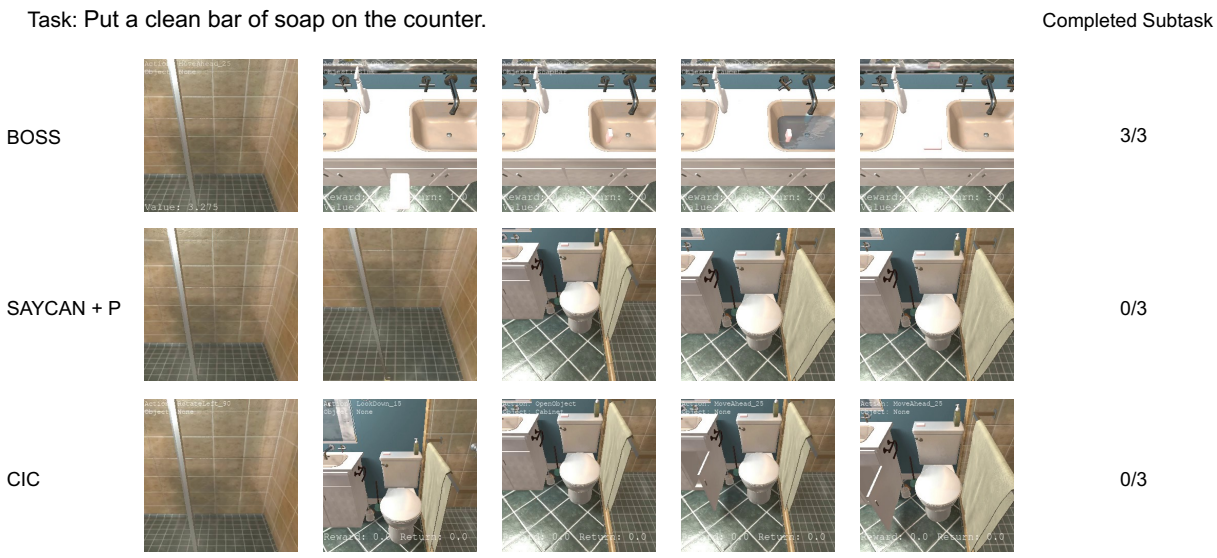
Predict the next skill correctly by choosing from the following skills: [SKILL 1 IN LIBRARY], [SKILL 2 IN LIBRARY], ...

Human: How would you [HIGH LEVEL TASK DESCRIPTION]?

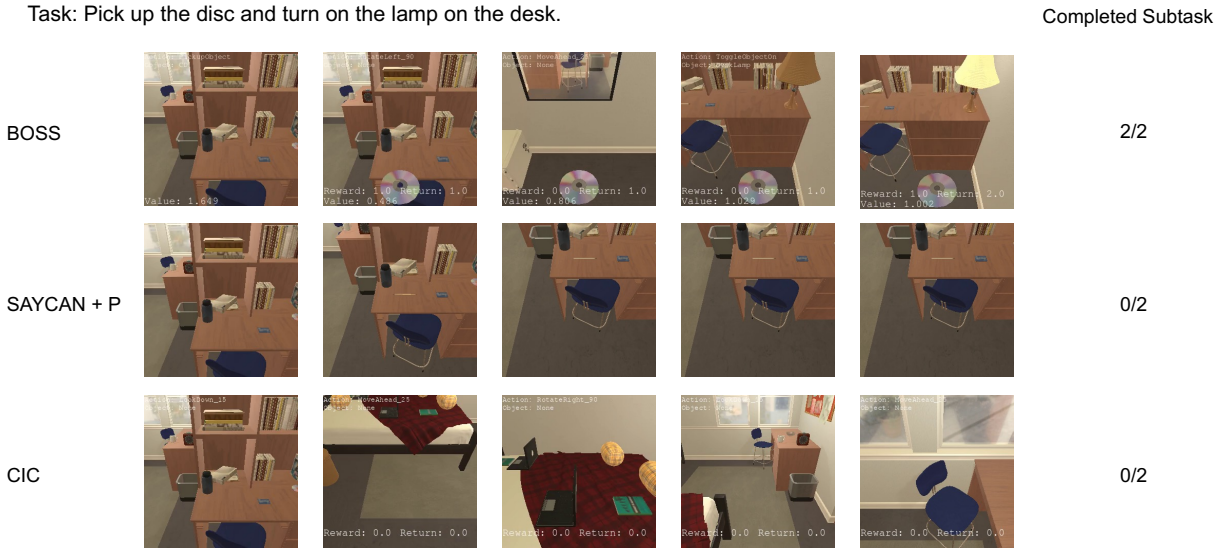
Robot: 1. [SKILL 1 EXECUTED SO FAR] 2. [SKILL 2 EXECUTED SO FAR] ... N. ____

F.2.6 ProgPrompt Implementation

ProgPrompt [331] converts natural language queries to code and executes the code on a real robot. After consulting with the authors, we converted the examples in our prompt to one suitable for ProgPrompt by converting task descriptions into a code representation by converting spaces into underscores, e.g., "Pick up the milk" into `def pick_up_the_milk()`. Then, to translate code commands into commands suitable for our pre-trained policy, we prompt ProgPrompt to output `pick_and_place(object, object)` style code commands that we convert into two separate pick and place natural language commands in the same format as the instructions used for pre-training the policy. We then execute these instructions on the real robot in sequence.



(a) Length 3 Task Example



(b) Length 2 Task Example

Figure F.4: Qualitative visualizations of zero-shot evaluation rollouts. See the plans SayCan+P generated for these two tasks at the top of Figure F.6.

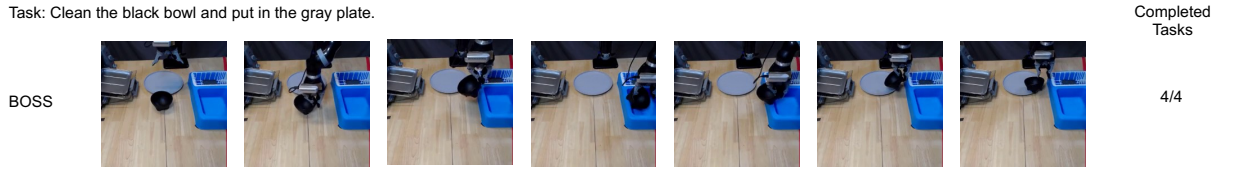


Figure F.5: Example of a BOSS rollout after skill bootstrapping on task 4: "Clean the black bowl and put it in the gray plate." BOSS is able to complete all 4 tasks in this rollout after performing skill bootstrapping.

F.3 Additional Results

F.3.1 ALFRED Results

SayCan Performance Analysis. Here, we analyze the performance of the SayCan baselines in great detail to determine *how* and *why* they perform poorly. SayCan errors occur for two reasons: (1) Planning errors in which the LLM fails to output the correct low-level instruction based on the high level task description, and (2) Policy execution errors in which the policy fails to execute the task correctly, given the correct instruction.

Qualitative examples of BOSS compared to SayCan+P and CIC are shown in Figure F.4, where we see that SayCan+P is unable to solve either task. Why is this? The first two plans in Figure F.6 correspond to the top two tasks in Figure F.4. As we can see, SayCan+P generated the correct first step but the policy failed to execute the skill as SayCan does not fine-tune policies in the environment. While Figure F.6 demonstrates that SayCan+P can make partial progress towards certain tasks, it relies on zero-shot LLM execution over fixed policies and therefore does not fine-tune the policies in the environment nor learn to *chain* them together so that the policy is robust enough to transition between skills in new settings.

We analyze the overall proportions of policy execution failures and planning failures for the SayCan baselines in Table F.1. We see that SayCan mostly fails at planning (57.5% of the time) while SayCan+P,

Task: Put a clean bar of soap on the counter. (Execution Fail)

Ground Truth	SayCan+P Generated Plan
1. Pick up the bar of soap. 2. Put the bar of soap in the sink, turn the water on and then off and then pick up the bar of soap. 3. Put the soap down in between the two sinks.	1. Pick up the bar of soap.

Task: Pick up the disc and turn on the lamp. (Execution Fail)

Ground Truth	SayCan+P Generated Plan
1. Pick up the disc on the desk. 2. Turn on the lamp on the desk.	1. Pick up the disc on the desk.

Task: Examine a bowl by the lamp. (Planning Fail)

Ground Truth	SayCan+P Generated Plan
1. Pick up the bowl on the desk. 2. Turn on the lamp.	1. Pick up the bowl on the desk. 2. Pick up the bowl on the desk.

Task: Put cooked apple slice on a counter. (Planning Fail)

Ground Truth	SayCan+P Generated Plan
1. Pick up the butter knife that is in front of the bowl on the counter. 2. Cut the apple that is in the garbage can into slices. 3. Put the knife in the garbage can. 4. Pick up a slice of apple that is in the garbage can. 5. Put the apple in the microwave and turn it on to cook, remove the cooked apple from the microwave. 6. Put the slice of apple on the counter to the right of the statue.	1. Pick up a slice of apple that is in the garbage can.

Figure F.6: Example plans from SayCan+P [7] evaluated on $EVAL_{INSTRUCT}$. SayCan+P errors mainly come from policy execution failures.

using BOSS' skill proposal mechanism, mainly fails at execution. Meanwhile, SayCan+PF performs similarly to SayCan+P, indicating that naïve fine-tuning does not greatly improve the success rate of the final plans.

SayCan+BOSS. Here, we test one more method which combines the advantages of top-down LLM planning methods like SayCan with BOSS' ability to enable agents to learn how to *chain together skills* directly in the target environment. We evaluate SayCan+BOSS, a baseline which breaks down high-level task instructions using SayCan and then issues the commands to BOSS agents after they have performed skill bootstrapping in the target environments. Results in the below table indicate that this baseline performs much better than BOSSalone, indicating that BOSS' LLM-guided skill bootstrapping enables it to learn robust policies that can even be combined with planners to better execute the given plans than naïve fine-tuning with SayCan+PF. Yet if there is no powerful LLM available at test time, BOSS alone still performs very well.

Table F.1: Comparison of SayCan and SayCan+P Methods

Method	Failure Rate (%)	
	Planning	Execution
SayCan	57.5	42.5
SayCan+P	4.2	95.8
SayCan+PF	5.0	95.0

	Evaluation Task Length			Average	
Method	Length 2	Length 3	Length 4	Return	Success
No Bootstrap	0.03 +- 0.02	0.05 +- 0.07	0.08 +- 0.09	0.03 +- 0.01	0.00 +- 0.00
CIC [178]	0.02 +- 0.02	0.25 +- 0.08	0.18 +- 0.07	0.11 +- 0.01	0.00 +- 0.00
SayCan [7]	0.06 +- 0.02	0.14 +- 0.00	0.10 +- 0.12	0.06 +- 0.00	0.00 +- 0.00
SayCan + P	0.08 +- 0.04	0.28 +- 0.00	0.20 +- 0.15	0.12 +- 0.01	0.00 +- 0.00
SayCan + PF	0.64 +- 0.06	0.49 +- 0.20	0.59 +- 0.02	0.57 +- 0.05	0.00 +- 0.00
BOSS (ours)	0.47 +- 0.12	0.59 +- 0.13	0.81 +- 0.13	0.57 +- 0.06	0.57 +- 0.14
SayCan+BOSS (ours)	**0.84 +- 0.16**	**0.87 +- 0.18**	**0.96 +- 0.13**	**0.84 +- 0.06**	**1.02 +- 0.12**

Table F.2: Full returns and success rates for real robot evaluation comparisons.

Task	ProgPrompt return	ProgPrompt success rate	BOSS return	BOSS success rate
1	1.6 ± 0.80	0.8	1.6 ± 0.8	0.8
2	1.0 ± 1.00	0.5	0.8 ± 0.75	0.2
3	0.9 ± 0.78	0.0	1.7 ± 1.1	0.1
4	2.0 ± 1.2	0.0	2.2 ± 0.98	0.2

F.3.2 Real Robot Results

We evaluate on 4 tasks, detailed below, in the environment setup shown in Figure F.5.

1. Clean the black bowl (length 2): (1) Pick up the black bowl, (2) put it in the sink.

2. Put the black bowl to the dish rack (length 2): (1) Pick up the black bowl, (2) put it in the dish rack.

3. Clean the black bowl and put it in the dish rack (length 4): (1) Pick up the black bowl, (2) put it in the sink, (3) pick up the black bowl, (4) put it in the dish rack.

4. Clean the black bowl and put it in the gray plate (length 4): (2) pick up the black bowl, (2) put it in the sink, (3) pick up the black bowl, (4) put it in the plate.

We report full results in Table F.2.

Algorithm 8 BOSS Algorithm

Require: Dataset $\mathcal{D}^L$ w/ language labels, LLM, Skill Library Z, Time limit T, max chain length M

Pre-train policy $\pi(a|s,z)$, value function $V(s,z)$ on $\mathcal{D}^L$ with offline RL. ▷ Section 8.3.1
while not converged **do**
    SkillBootstrapping(V, Z, LLM, π, $\mathcal{D}^L$, M, T) ▷ Section 8.3.2

procedure SkillBootstrapping(V, Z, LLM, π, $\mathcal{D}^L$, M, T)
    $s_1 \leftarrow$ Reset environment
    `RolloutData` $\leftarrow []$
    $z \leftarrow$ sample from discrete distribution with probs $\left[V(s,z_1), V(s,z_2), ..., V(s,z_{|Z|})\right]$.
    $i \leftarrow 0$
    `Success` $\leftarrow$ True
    while $i < M$ and `Success` **do** ▷ If a rollout fails, break the loop.
        $i \leftarrow i + 1$
        (`Success`, τ) $\leftarrow$ Rollout $\pi(\cdot|s,z)$ in Environment for at most T steps.
        Add τ to `RolloutData`
        if `Success` **then**
            $z \leftarrow$ SampleNextSkill(LLM, `RolloutData`, Z)
    UpdateBufferAndSkillRepertoire($\mathcal{D}^L$, `RolloutData`, LLM)
    Train π, V on $\mathcal{D}^L$ with offline RL.

procedure SampleNextSkill(LLM, `RolloutData`, Z)
    `AllSkills` $\leftarrow$ extract all skill annotations from Z.
    `SkillChain` $\leftarrow$ extract executed primitive skills from `RolloutData`.
    `Prompt` $\leftarrow$ construct prompt from `AllSkills`, `SkillChain`. ▷ Prompt in Figure F.2.
    $([\hat{z}_1, ..., \hat{z}_N], [p_1, ..., p_N]) \leftarrow$ Sample N text generations from LLM(`Prompt`) with average token probabilities $p_1, ..., p_N$.
    Find closest match in Z to each of $\hat{z}_1, ..., \hat{z}_N$ in embedding space ▷ Embedding model: `all-mpnet-base-v2` from Reimers and Gurevych [298].
    $z \leftarrow$ sample the matches in Z from categorical distribution with parameters $p_1, ..., p_N$.
    return z

procedure UpdateBufferAndSkillRepertoire($\mathcal{D}^L$, `RolloutData`, Z, LLM) ▷ See Appendix F.2.3 for details.
    $\tau_1, ..., \tau_k \leftarrow$ extract primitive skill trajectories from `RolloutData`.
    for τ_i in $\tau_1, ..., \tau_k$ **do**
        $\mathcal{D}^L \leftarrow \mathcal{D}^L \cup \{\tau_{i,z_i}\}$ ▷ Add trajectory to $\mathcal{D}^L$ with annotation z_i.
    $\tau_{1:k} \leftarrow$ concatenate all trajectories together
    $z_{LLM,1:k} \leftarrow LLM(\tau_{1:k})$ assign name by asking LLM summarize annotations of $\tau_{1:k}$. ▷ See Appendix F.1.2 for prompt.
    $z_{concat,1:k} \leftarrow$ "$\{z_1\}.\{z_2\}...\{z_k\}.$' ▷ Assign another label for the trajectory by concatenating primitive skill annotations.
    $\mathcal{D}^L \leftarrow \mathcal{D}^L \cup \{\tau_{LLM,1:k}, \tau_{concat,1:k}\}$ ▷ Add to $\mathcal{D}^L$ with annotation $z_{LLM,1:k}$ and $z_{concat,1:k}$.
    Add $z_{LLM,1:k}$ as a new skill to Z.

Appendix G

REWIND

G.1 Implementation Details

This section introduces implementation details for ReWiND in terms of the datasets, reward model, policy training, and online RL.

G.1.1 ReWiND Implementation

Full pseudocode for ReWiND is listed in Algorithm 9. Individual implementation details follow.

G.1.1.1 Open-X Dataset

Below we list details of the OXE subset, $\mathcal{D}_{\text{open-x}}$, used for training the reward model $R_\psi(o_{1:t}, z)$ (mentioned in Section 10.3.1.1).

We select a subset of datasets from the Open-X Dataset [67]. The subset includes Bridge-V2 [362], BC-Z [151], Fractal [39], CLVR Jaco Play [76], Berkeley Autolab UR5 [53], Berkeley Fanuc Manipulation [418], CMU Stretch [20, 239], Stanford Hydra [26], UCSD Kitchen [384], Austin BUDS [419], and Austin Sirius [206]. These datasets were selected for their high-quality, task-oriented manipulation trajectories (i.e., no play data or extremely high-level annotations). These datasets provide around 350k trajectories and 58k total unique task annotations. To ensure meaningful trajectories for training the ReWiND reward model,

Algorithm 9 ReWiND Algorithm, Section 10.3.

Require: Demo dataset $\mathcal{D}_{\text{demos}}$, Pre-trained LLM, Open-X subset $\mathcal{D}_{\text{open-x}}$, Reward Model $R_\psi(o_{1:t}, z)$, Policy π. $\mathcal{D}_{\text{demos}}$ includes video trajectories $o_{1:t}$ and language embedding z.

/* *Train the Reward Model* Section 10.3.1 */
RewardModelTraining($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, $\mathcal{D}_{\text{open-x}}$)
/* *Policy Pretraining* Section 10.3.2 */
OfflinePolicyPretraining($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, π)
/* *Learn New Task Online* Section 10.3.2 */
OnlineRL(z_{new}, $R_\psi(o_{1:t}, z)$, π)

procedure RewardModelTraining($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, $\mathcal{D}_{\text{open-x}}$)
    Augment instruction labels with LLM
    Sample a video clip and annotation $o_{t_1:t_2}, z$ from $\mathcal{D}_{\text{demos}}$ or $\mathcal{D}_{\text{open-x}}$.
    Choose to keep the original video or perform RewindAugmentation.
    if perform RewindAugmentation **then**
        $o^{\text{rewound}} \leftarrow \text{RewindAugmentation}(o_{t_1:t_2})$
        Optimize $R_\psi(o_{1:t}, z)$ with $\mathcal{L}_{\text{rewind}}(o^{\text{rewound}}, z)$ ▷ Equation (10.2)
    else
        Sample a different video clip $o^{\text{other}}_{t_1':t_2'}$
        Optimize $R_\psi(o_{1:t}, z)$ with $\mathcal{L}_{\text{progress}}(o_{t_1:t_2}, z, o^{\text{other}}_{t_1',t_2'})$ ▷ Equation (10.1)

procedure OfflinePolicyPretraining($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, π)
    Relabel $\mathcal{D}_{\text{demos}}$ with $\hat{r}^{\text{off}}$ coming from $R_\psi(o_{1:t}, z)$. ▷ Equation (10.4)
    Train π with offline RL on relabeled $\mathcal{D}_{\text{demos}}$.

procedure OnlineRL($R_\psi(o_{1:t}, z)$, π)
    For every rollout label the trajectories with $\hat{r}^{\text{on}}$ from $R_\psi(o_{1:t}, z)$. ▷ Equation (10.5)
    Optimize π with online RL Algorithm

procedure RewindAugmentation($o_{t_1:t_2}$) ▷ Section 10.3.1.2
    Sample random split point i between t_1 and t_2.
    Sample # frames to rewind for, k
    Reverse $o_{i-k:i}$ and concat with $o_{t_1:i}$
    Return $[o_{t_1:i-1}, o_{i:i-k}]$

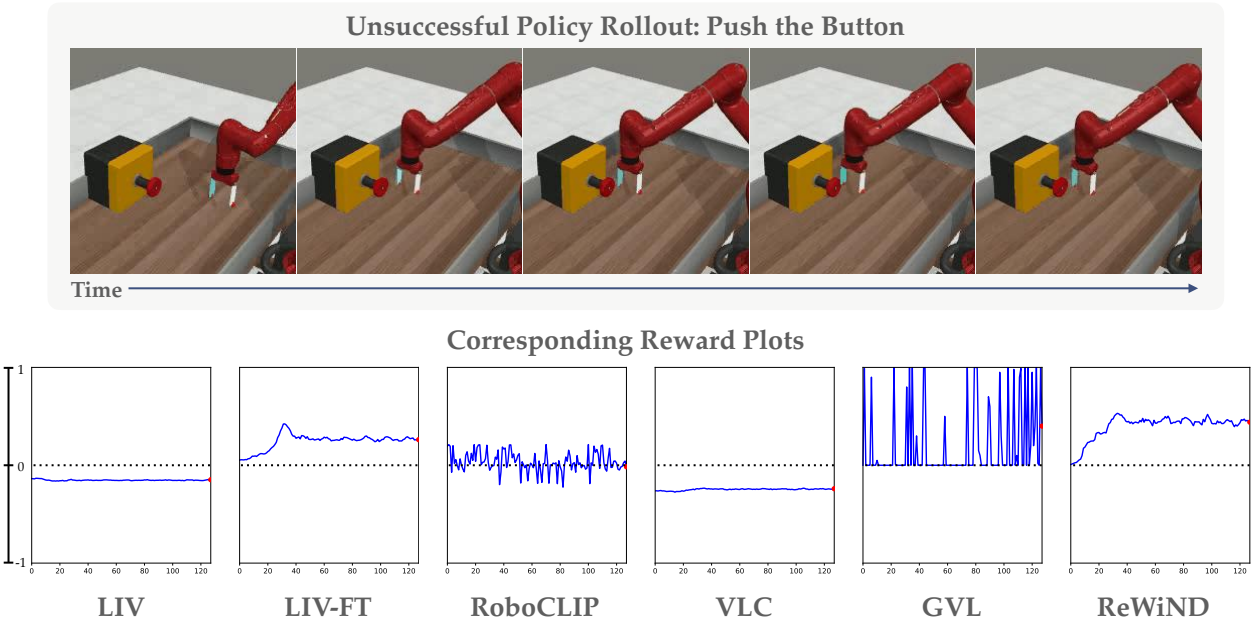


Figure G.1: Unsuccessful policy rollout for the "Push the Button" task in Meta-World and its corresponding rewards below it. ReWiND predicts calibrated rewards that reflect better partial progress when the policy gets stuck near the button.

we postprocess the data to remove trajectories with less than 5 timesteps. We subsample the videos in the datasets to 16 frames for reward model training, as we did not see a noticeable benefit from training it with longer videos.

G.1.1.2 Reward Function

We picture the overall architecture of the reward function in Figure G.2. We encode input images with the pre-trained DINO-v2 base model (86M params) with 768 embedding size. Similarly, we encode language with the pre-trained ALL-MINILM-L12-V2 model with a 384 embedding size. We project image and language embeddings to 512 dimensions with a single linear layer. We treat the language embedding as a single input token and we evenly downsample DINO-v2 image embeddings for every observation sequence to 16 frames.

The cross-modal sequential aggregator takes these tokens as input and produces a per-image embedding used by an MLP to produce per-timestep rewards. The cross-modal sequential aggregator is a causally

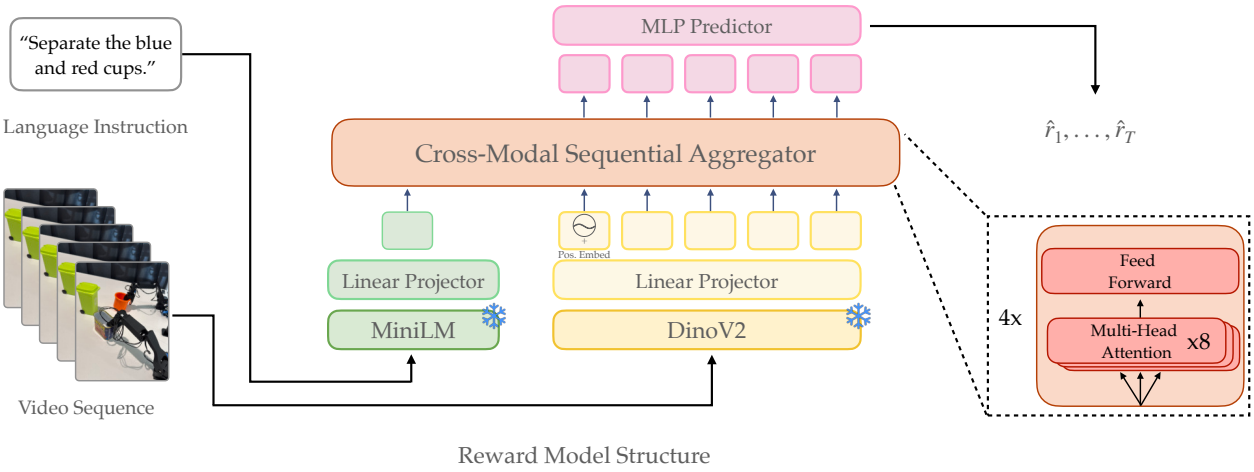


Figure G.2: **ReWiND's Reward Model Architecture**. It's composed of frozen language and image input embeddings projected to a shared hidden dimension of 512. These embeddings are treated as input tokens to the cross-modal sequential aggregator transformer composed of 4 causally masked transformer layers composed of 8 multi-head attention blocks each. Per-timestep embeddings for each input observation are fed into an MLP to predict rewards for each timestep.

masked transformer (`PyTorch nn.TransformerEncoder`) composed of 4 layers, each with 8 heads with a combined hidden dimension of 2048. We add a learnable positional embedding to only the first frame of the video sequence embedding. In the ReWind reward function training phase, we trained 2k steps for Meta-World and 10k steps for Real-World robot experiments, with a batch size of 1024. Each batch includes 80% data from $\mathcal{D}_{\text{open-x}}$ and 20% target environment data from $\mathcal{D}_{\text{demos}}$. Each video in the batch has an 80% probability of having video rewind augmentation, and independently, a 20% percent probability of having a mismatched video-language pairing with 0 progress target (see Section 10.3.1). In order to better policy execution videos that look *close to success*, 10% of the rewound videos will only have their last 3 frames rewound. No extensive tuning was performed on these per-sample rewind and mismatch probabilities; they were heuristically chosen during initial small-scale experimentation and then fixed for all experiments.

G.1.1.3 Policy Training

Specific architectural and training details are discussed per-environment in the corresponding sections Appendix G.2.2 and Appendix G.3.2. Below we talk about high level algorithmic details for policy training along with shared implementation details across environments.

Policy Input. Similar to the reward model, we condition the policy on frozen pre-trained image and language embeddings: DINO-v2-base image embeddings (86M params, 768 embedding size) [269] along with ALL-MINILM-L12-V2 language embeddings of size 384 from the Sentence Transformers python package [298].

Offline RL. We use Implicit Q Learning (IQL) [171] as prior work found it performant and easy to tune for robot manipulation with action-chunked policies [407, 405, 373]. IQL trains on in-distribution (s, a, s', r, a') tuples from the dataset, avoiding policy-sampled a' to ensure the Q- and value functions accurately reflect returns restricted to dataset actions. The value function is optimized with expectile regression, controlled by a hyperparameter τ: $\tau = 0.5$ recovers mean squared error, while $\tau \to 1$ yields a more optimistic estimate, helping the value function "stitch" together distant rewards in sparse settings. The policy is trained via advantage-weighted regression [277], maximizing

$$e^{\beta(Q(s,a)-V(s))} \log \pi(a|s),$$

where β is a temperature hyperparameter controlling how "spiky" the policy loss is. To prevent numerical instability, the exponential term is capped at a maximum value in practice (for us, this is 100).

Online RL. We use a custom soft-actor critic (SAC) [122] implementation initialized with the pre-trained policy from offline RL along with the Q and target Q functions. We follow best practices from recent offline-online RL fine-tuning work [416, 24], namely:

- 5-10 critics instead of 2, with random sampling of critics
- LayerNorm in the critic and possibly LayerNorm in the policy
- A higher update-to-data ratio in the critics
- "Warm-starting" online RL by running with the frozen pre-trained policy for the first few thousand environment steps [416, 356]
- Possibly sampling offline pre-training data at a 50% ratio during online RL
- Removing the SAC entropy term from the target critic

We found that by default, efficient offline-online learning algorithms did not work very well "out of the box" for learning *new* tasks on our real robot. This is perhaps because they focus specifically on offline-online fine-tuning on the same task while we are trying to learn new tasks, or perhaps due to additional challenges of real-robot RL. Therefore, we make some per-environment design decisions for online RL detailed in the respective environment training sections.

G.2 MetaWorld Experiments

G.2.1 Simulation Setup

Training/Eval Task Selection. We manually select 20 training tasks from MT50 benchmark in the Metaworld environment. These tasks are used for both reward model training and policy pre-training. The training tasks include: `Button-Press`, `Button-Press-Topdown-Wall`, `Coffee-Pull`, `Dial-Turn`, `Door-Open`, `Door-Unlock`, `Drawer-Close`, `Faucet-Open`, `Handle-Press`, `Handle-Pull-Side`, `Peg-Insert-Si` `Pick-Place`, `Plate-Slide`, `Plate-Slide-Back-Side`, `Push`, `Reach`, `Stick-Push`, `Stick-Pull`, `Window-Open`, `Hand-Insert`.

We also choose another 17 tasks from the MT50 benchmark for reward model evaluation and 8 of tasks are selected for downstream policy finetuning.* The evaluation tasks include `Window-Close`, `Sweep-Into`, `Soccer`, `Reach-Wall`, `Push-Back`, `Plate-Slide-Side`, `Plate-Slide-Back`, `Pick-Place-Wall`, `Handle-Pull`, `Handle-Press-Side`, `Faucet-Close`, `Door-Lock`, `Door-Close`, `Coffee-Push`, `Coffee-Button`, `Button-Press` `Button-Press-Topdown`. These tasks are visually similar to the training tasks, but the tasks are different. The 8 tasks used for downstream policy training are `Window-Close`, `Reach-Wall`, `Handle-Pull`, `Coffee-Button`, `Button-Press-Wall`, `Door-Lock`, `Handle-Press-Side`, `Sweep-into`.

Environment Details. We use Metaworld [394] with the default 3rd-person camera viewpoint, pictured in Figure G.3, and also 4-dimension proprioception input (x, y, z, gripper). The policy action space is the default one from Metaworld represented as a 4-dimensional relative action space for ($\Delta x, \Delta y, \Delta z$, gripper). Unlike the Metaworld environment setups in prior reward learning papers, we do not include goal/ground truth state information. We also terminate the environment on success. Both of these choices were made to mimic a real-world robot learning setup. The time horizon of each episode is limited to 128 steps.

Figure G.3: Example camera viewpoint in Metaworld.

G.2.2 Training Details

For $\mathcal{D}_{\text{demos}}$, we select 20 tasks from the MT-50 benchmark. Each task consists of one human-labeled annotation, four augmented annotations, and five optimal trajectories produced by the MetaWorld built-in planner. We render images at the default resolution of 640x480, centercrop to 224x224 and embed the image with DINOv2 encoder.

*These 17 tasks were chosen for sharing at least some characteristic with a training task.

Figure G.4: **Real World Bimanual Robot Setup.** Our real-world setup consists of a top-down and side camera mounted to a table where two Koch v1.1 low-cost arms are mounted. This setup allows us to perform bimanual tasks and easily collect data with another pair of low-cost "leader" arms mounted to the same table.

We pre-train the policy with IQL [171] for 100K steps with learning rate 0.001, gamma 0.99. We use a three layer MLP of size $[768, 512, 256]$ for both the policy and value function network. The training procedure is described in (**??**)

For the various hyperparameters for online policy learning we used in MetaWorld as described in Section G.1.1.3. We use 10 critics and sample 2 of them during training, LayerNorm in both the critic and policy, and an update-to-data ratio of 4 for the critics. We are not sampling from offline pre-training data during online nor are we training the target critic with the entropy term so the implementation is identical to Warm-Start RL [416]. We warm-start online RL for 4000 steps.

G.3 Real Robot Experiments

G.3.1 Robot Experiment Setup

We use the Koch1.1 bimanual arm setup for data collection and learning [47].[†] Altogether, four total arms (2 for data collection) cost ∼$1000, letting us demonstrate ReWiND enables real-world online RL of new tasks even with very low-cost hardware and noisy control. The observations consist of RGB images from

[†] https://github.com/jess-moss/koch-v1-1

a Logitech C930e top camera and side camera (pictured in Figure G.4). We control the robot with absolute joint position control at a frequency of 30Hz. We collect a small dataset of 10 demonstrations over 20 tasks, and then use 5 demos per-task for the reward function. We found the offline-trained policy to be the primary bottleneck to optimizing rewards in unseen tasks, so we used 10 demos per-task for offline policy training.

G.3.2 Real Robot Training Details

We use a small, instruction-tuned, open-source LLM, `Mistral-7B-Instruct-v0.3` [153], to generate 9 additional instructions for each task for instruction augmentation.

For the small dataset in real robot experiments, we manually choose 15 tasks in the Koch tabletop setting, and each task includes 5 trajectories and 10 annotations. The evaluation set is 5 other random tasks, which are irrelevant with the tasks in the small dataset. We use this evaluation set for offline metrics and validating various design choices.

Unlike the MetaWorld experiments that use an MLP-based policy, we use an action-chunked policy with temporal ensembling for the real robot. We found chunking to lead to more stable bimanual manipulation on the Koch arms. We implement the action chunking with a Transformer policy that predicts 60 actions at each timesteps corresponding to 2 seconds of actions. We also implement a Transformer-based critic. During rollouts, we then use temporal ensembling [410]. Here, the current action is ensembles with the last 60 timesteps' predictions according to an exponential weighting scheme $w_i = \exp(-m * i)$, where we use $m = 0.01$ or $m = 0.1$ depending on the task. We found $m = 0.1$ to work well for tasks requiring grasping solid objects as it weights recent actions more heavily, necessary for ensuring the policy actually commits to the grasp, and $m = 0.01$ to work well for non-grasping tasks as it results in a smoother policy

We train each policy for 20k steps offline on our offline dataset using IQL with AWR for policy extraction. We train using a batch size of 256, use 5 critics, and subsample 2 critics at each training step. We use

LayerNorm only in the critics as we found that LayerNorm in the action-chunked policy could potentially hurt RL performance. We also warm-start online RL for 3000 steps.

Then, we train the policy online as described in Section 10.3.2. During online training, we collect a single rollout, add it to the replay buffer, and then train for 75 gradient steps. Then, at each gradient step, we sample our buffer such that 50% is the offline training data, 25% is online failure trajectories, and 25% is online successful trajectories. This sampling approach helps upsample successful online trajectories. For every actor gradient step, we do 5 critic update steps to more quickly train the critic online. We train online for 50k actor steps, which takes approximately 1 hour as there is no minimal waiting time for policy training due to a threaded implementation that trains the policy while the last iteration's policy checkpoint is used for rollouts. This parallelization nearly doubles the rate at which we are able to collect policy rollouts.

During real-world policy rollouts, it is important for the robot to take safe actions that will not crash into other objects or the table. However, we found that when regularizing the policy's KL divergence against a max-entropy prior as is the case in the entropy maximization objective in standard SAC [122], the growing entropy term would cause the policy to produce largely random actions. Therefore, we regularize against the pretrained policy's distribution to encourage reasonable behaviors throughout the process of learning, similar to the SAC update rules from Pertsch, Lee, and Lim [282] and Zhang et al. [402]. Thus the π and Q updates follow:

$$\pi \leftarrow \max_{\pi} \mathbb{E}_{\pi}\Big[Q(o,z) - \alpha \underbrace{\mathrm{KL}(\pi(\cdot|o,z) \;||\; \pi_{\text{pretrained}}(\cdot|o,z)}_{\text{pre-trained policy guidance}}\Big] \tag{G.1}$$

$$Q \leftarrow \min_{Q} Q(o,z) = r + \gamma\Big[Q(s',z',d') - \alpha \mathrm{KL}\left(\pi(\cdot|o,z) \;||\; \pi_{\text{pretrained}}(\cdot|o,z)\right)\Big] \tag{G.2}$$

We set α in both equations to a fixed value of 10.0 on tasks where grasping solid objects is not required. For others, we set it to 20.0 to ensure the policy doesn't degenerate from its grasping action early in

training. We found that lower KL penalties could result in the policy falling into locally optimal but globally suboptimal behaviors, such as moving a cup with the arm instead of actually picking it up.

G.3.3 Real Robot Tasks

We collected 10 demos per-task over 20 tasks on the Koch arms. We train the reward function on 5 demos per-task and the policy on 10 demos per-task. We list these training tasks below.

`Move the orange cup from the left to the right`, `Move the orange cup from the right to the left`, `Put the orange cup on the red plate`, `Put the red cup on the red plate`, `Separate the blue and red cups`, `Fold the blue towel`, `Open the green trash bin`, `Open the blue trash bin`, `Throw the banana away in the green trash bin`, `Throw the banana away in the blue trash bin`, `Put the red marker in the red trash can`, `Put the pink marker in the green trash can`, `Put the blue tape in the box on the left`, `Put the banana in the box`, `Put the orange cup in the box`, `Put the blue cup on the red plate`, `Separate the orange and blue cups`, `Open the red trash bin`, `Throw the banana away in the red trash bin`, `Put the red tape in the box on the right`.

In addition, we present rollouts of the five online tasks in Figure G.5. We also provide additional descriptions of these tasks below:

- `Separate the blue and orange cups`: the robot must separate the two cups in the middle
- `Fold the blue towel`: the robot must fold the towel in half.
- `Open the red trash bin`: the robot is surrounded by clutter compared to the training data above and must open the trash bin
- `Put the orange cup in the red plate`: the robot picks an orange cup and must place it on a plate that is further away from the training data distribution

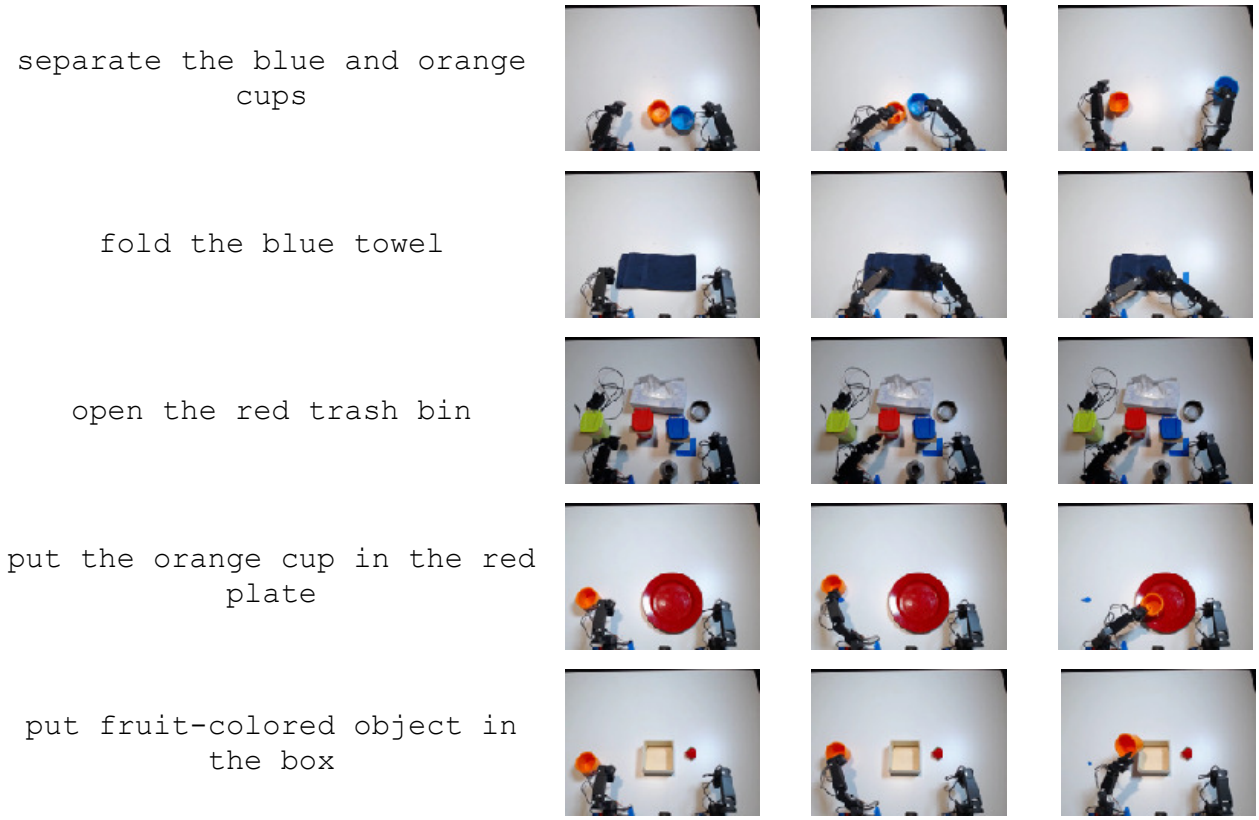


Figure G.5: We present rollouts for the 5 tasks we use for online RL. The first two tasks are in-distribution to the policy, while the latter 3 tasks are out-of-distribution with respect to visual, spatial, or semantic generalization.

- `Put fruit-colored object in the box`: we refer to a “fruit-colored" object to test the robot’s ability to handle semantic generalization.

G.4 Additional Results

G.4.1 Additional Metaworld Reward Analysis

In Figure G.6 we plot the confusion matrices of different reward models on training tasks in addition to the evaluation task plots of Metaworld in Figure 10.4. LIV, RoboCLIP and GVL are not pretrained or fine-tuned on the etraining tasks while VLC, LIV-FT and ReWiND are. We can see both ReWiND w/ OXE data $\mathcal{D}_{\text{open-x}}$ and ReWiND w/o OXE data $\mathcal{D}_{\text{open-x}}$ are the best, having the clearest disparity between the diagonal and

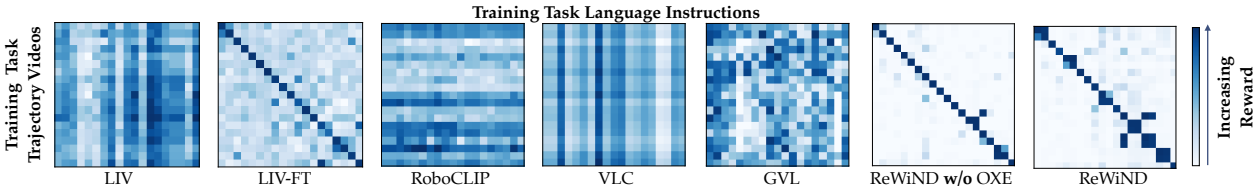


Figure G.6: **Metaworld Reward Confusion Matrix on 20 Training Tasks.** For each training task, we compute rewards for all combinations of demonstration videos and language descriptions. ReWiND produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions.

off-diagonal elements. LIV-FT also works well with a diagonal-heavy matrix. However, its disparity is not as clear as ReWiND.

G.4.2 MetaWorld Sample Efficiency Results

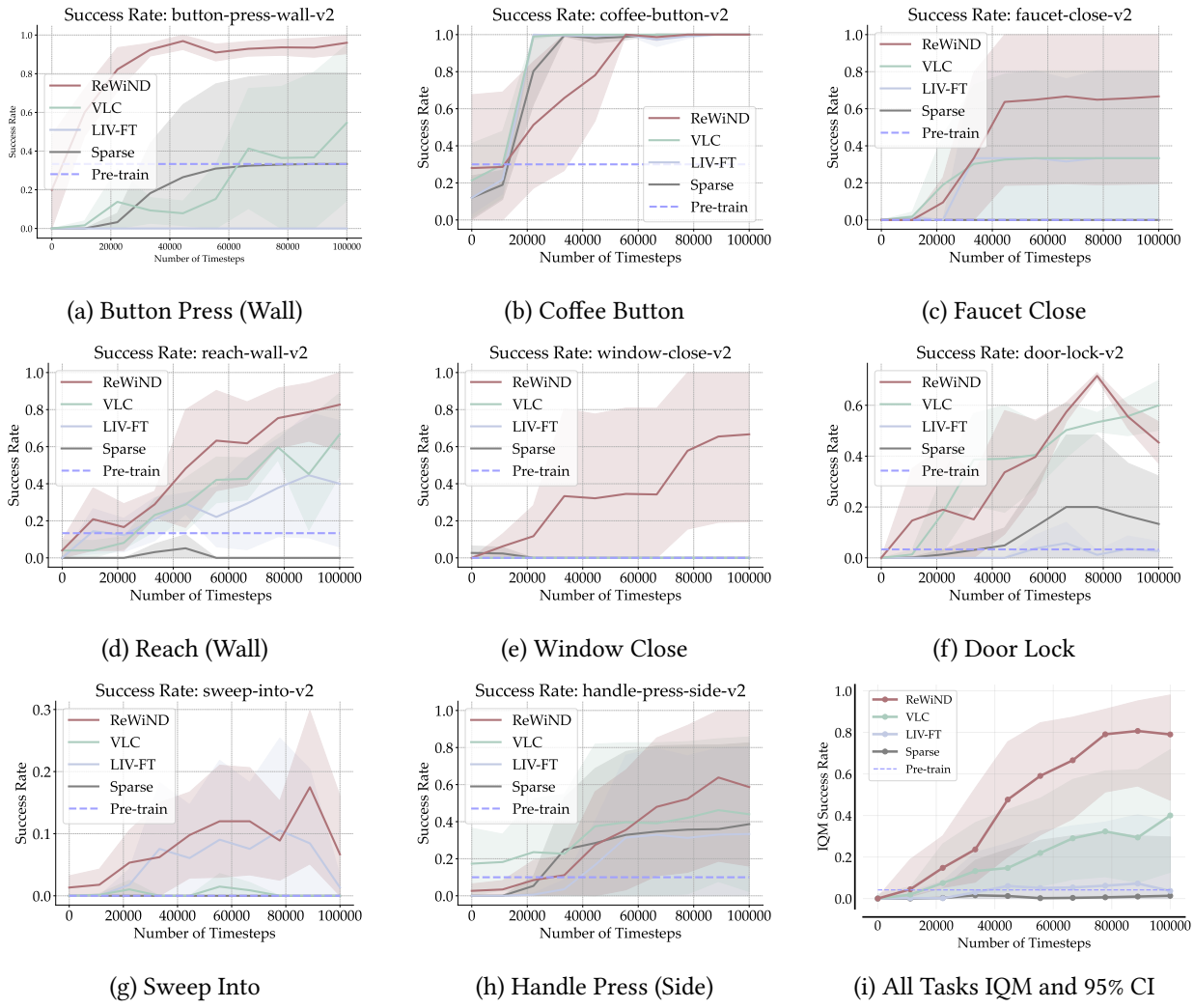
In this section, we analyze the sample efficiency of ReWiND against baselines in Metaworld. Figure G.7 plots the learning curves for all downstream policy training tasks. Each panel corresponds to one specific task. And Figure G.7i displays the average of all 8 downstream tasks we used for policy fine-tuning. We can see from the average IQM plot that ReWiND achieves higher success rate than other baselines with the same number of timesteps and ReWiND is generally more sample-efficient at any timestep.

G.4.3 Real-World Reward Analysis

We evaluated the performance of ReWiND in Metaworld in Section 10.4.1. In this section, we analyze how ReWiND works with real-world data. It can be seen from Table G.1 that ReWiND has the highest Spearman's rank correlation (ρ) and Pearson's rank correlation (r) among all reward models. Also, in fig. G.8 and fig. G.9, ReWiND has the best alignment between true-paired video

Table G.1: **Evaluation Metrics on Real-world Unseen Tasks:** Comparsion between reward models in real-world unseen tasks with rank correlation ρ and r.

Model	LIV	LIV-FT	RoboCLIP	VLC	GVL	ReWiND
$\rho \uparrow$	0.22	-0.18	0.04	0.20	0.57	**0.91**
$r \uparrow$	0.23	-0.13	0.04	0.19	0.52	**0.91**



(a) Button Press (Wall) (b) Coffee Button (c) Faucet Close

(d) Reach (Wall) (e) Window Close (f) Door Lock

(g) Sweep Into (h) Handle Press (Side) (i) All Tasks IQM and 95% CI

Figure G.7: **Metaworld success curves.** Task-level success rate learning curves plotting mean and shaded standard deviations. The bottom right figure plots the overall average across all tasks in terms of IQM and 95% confidence intervals.

and language instruction in both training tasks and unseen tasks, displaying strong generalization in new tasks. Note that LIV, GVL, and RoboCLIP are not trained on these training tasks as they are zero-shot models.

G.5 Ablation Study and Additional Analysis

In this section, we perform an ablation study of various ReWiND reward components and also perform additional analysis of results not included in the main paper.

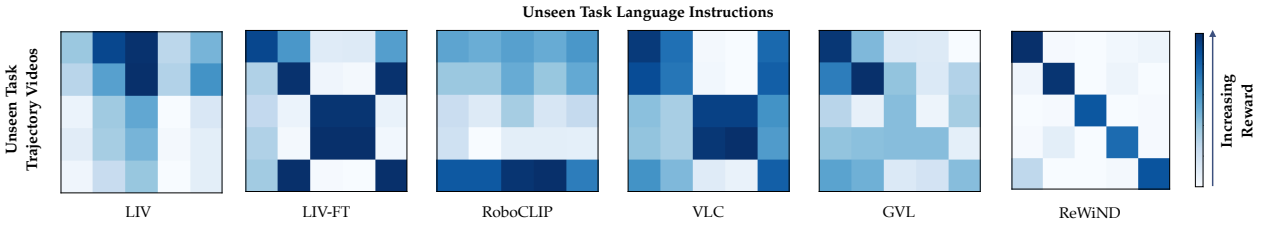


Figure G.8: **Real-world Koch Reward Confusion Matrix on 5 Unseen Tasks.** For each unseen task, we compute rewards for all combinations of demonstration videos and language descriptions. ReWiND produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions.

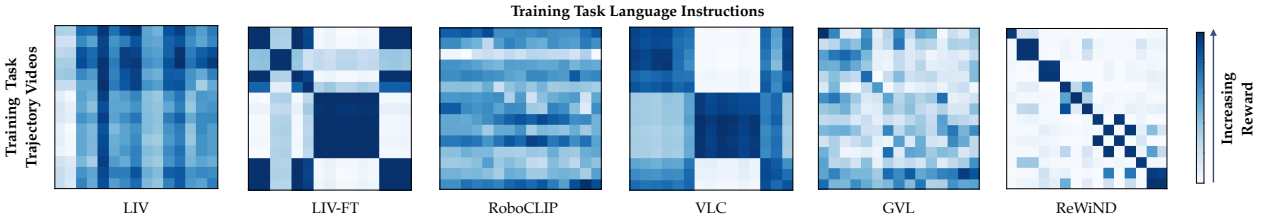


Figure G.9: **Real-world Koch Reward Confusion Matrix on 15 Training Tasks.** For each training task, we compute rewards for all combinations of demonstration videos and language descriptions. LIV-FT, VLC and ReWiND are pretrained or fine-tuned with these training task while LIV , GVL and RoboCLIP are not

G.5.1 Ablation Study

We perform a thorough ablation study of ReWiND regarding how specific design choices influence demonstration reward alignment, policy rollout ranking, and input robustness metrics introduced in Section 10.4.1. We ablate: instruction generation and video rewinding (Section 10.3.1.2); using OXE data (Section 10.3.1.1); the need for target environment data $\mathcal{D}_{\text{demos}}$; and finally, the use of first frame vs. full frame positional embeddings on the input observation sequence $o_{1:T}$ in the cross-modal sequential aggregator (Section 10.3.1.3).

Overall, the original ReWiND model performs the best on most metrics, and is often the second best on other metrics where it is not the best. We go through and analyze each ablation below:

- $-$**Targ. Env Data** (i.e., no $\mathcal{D}_{\text{demos}}$, only $\mathcal{D}_{\text{open-x}}$) does not perform well on training demonstration ρ alignment in Table G.2(a) and is also unable to properly distinguish failed, almost successful, and successful policy rollouts in Table G.2(b). Yet, it can perform well in terms of input robustness as it still maintains OXE data.

Table G.2: **Ablation Study:** subtracting (−) and adding + various ReWiND components on training and evaluation task (a) demo reward alignment, evaluation task (b) policy rollout ranking order and reward difference, and evaluation task (c) input robustness.

Model	(a) Demo Reward Alignment		(b) Policy Rollout Ranking		(c) Input Robustness	
	Train Demos $\rho \uparrow$	Unseen Demo $\rho \uparrow$	Rew. Order $\rho \uparrow$	Rew. Diff. $\uparrow$	Avg. $\rho \uparrow$	ρ Variance $\downarrow$
Original ReWiND	**1.00**	0.79	**0.82**	**0.41**	0.74	0.04
− **Targ. Env Data**	0.55	0.77	0.18	0.08	**0.78**	0.04
− **Open-X Subset**	**1.00**	0.64	0.76	0.39	0.55	0.03
− **Video Rewind**	**1.00**	0.69	0.56	0.27	0.66	**0.02**
− **Instr. Generation**	**1.00**	0.66	0.62	0.30	0.52	0.07
+ **Full Pos. Embeds**	0.99	**0.85**	0.71	0.33	**0.78**	0.06

- Meanwhile, (−**Open-X Subset)** suffers in terms of unseen task reward alignment (a). It also suffers in terms of input robustness (c) as OXE data helps with seeing more language instructions.
- −**Video Rewind** performs poorly on (b) policy rollout ranking metrics against the original model, demonstrating that **rewinding significantly helps with properly distinguishing failed policy rollouts**, as designed.
- −**Instruction Generation** performs poorly on (c) language input robustness metrics, highlighting how LLM instruction generation helps the reward model be more robust to diverse inputs.
- +**Full Pos. Embeds** actually performs *better* on unseen demo ρ in (a), but performs worse in (b) policy rollout ranking metrics. This performance drop likely occurs because the reward model becomes slightly overfit to just predicting increasing rewards (by using the positional embeddings to cheat) regardless of what the input observation video is. ReWiND's model as presented in the main text uses just first-frame positional embeddings for this reason (Section 10.3.1.3).