Acting with Language

Mohit Shridhar

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Dieter Fox, Chair

Luke Zettlemoyer

Leslie Kaelbling

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

Acting with Language

Mohit Shridhar

Chair of the Supervisory Committee:
Dieter Fox
Computer Science and Engineering

How can we imbue robots with the ability to achieve arbitrary goals in novel environments? Language provides a natural interface for guiding robots and abstracting the complexities of the physical world. Previous attempts to guide robots with language often rely on human-designed intermediate representations, such as object detections, categories, poses, and symbolic states. These representations struggle to represent everyday objects, such as deformable shirts, coffee beans, ropes, and cherry stems.

One solution that does not require human-designed representations is end-to-end deep learning, which directly maps camera observations to robot actions. While learning approaches are vastly more expressive than traditional methods, they are severely bottlenecked by the lack of training data in robotics. Training a simple policy could take months of data collection and is not scalable. However, robot data includes spatial symmetries and other structural priors that can be utilized to efficiently learn policies for a wide range of tasks.

In this thesis, we present various methods for using language to guide robot actions through end-to-end learning. First, we present ALFRED, a large-scale dataset and benchmark for evaluating agents that follow language instructions in partially-observable household environments. Next, we introduce CLIPort and PerAct, two language-conditioned manipulation frameworks that aim to replicate the success of pre-training large models from vision and language in robotics. These frameworks use spatial priors to efficiently learn action representations from limited data. Lastly, we discuss ALFWorld, a framework for learning "textual policies" in interactive text games, thereby avoiding the visual and physical complexities of interacting with embodied environments. We conclude with a discussion on counterpoints, limitations, and potential future directions for scaling-up robot-learning and butler robots.

Acknowledgments

First and foremost, I express my deepest gratitude to my parents, Mangala and Shridhar, who have been a constant source of love and support throughout my life. They encouraged me to pursue my interest in science since I was young and never doubted my unconventional path of pursuing a career in research. From winning my first science-fair project to working during vacations to submit papers, they have always been there for me.

I am immensely grateful to my advisor, Dieter Fox, for his guidance and support. I came to UW with the hope of working on RoboNLP, and I have had the freedom to work on RoboNLP for the past 5 years, thanks to his trust and encouragement. Despite Dieter's prior expertise being in state-estimation and sim-to-real methods, which is opposite to my research direction of training end-to-end methods in the real-world, he provided insightful feedback, had extended discussions, established collaborations, polished my experiments, and encouraged me to follow my hunches. His open-mindedness, drive, wit, humility, and unmatched athleticism have been an inspiration and continue to inspire me. I feel extremely lucky to have found such a supportive advisor.

I would not have started my research career without the support and help of David Hsu. David mentored me as an undergrad in NUS and played a crucial role in shaping my research interests and taste. He taught me how to pick problems, present them, and delve deeper into them. Without his constant encouragement and life advice, I would not have been able to join a PhD program or had the courage to carve out my own research path.

I am also thankful to Luke Zettlemoyer for his continued guidance during my PhD. Without his support, several of my projects would not have seen the light of day. I extend my gratitude to Leslie Kaelbling and Blake Hannaford for being on my committee and providing insightful feedback for my generals presentation. Leslie has been supportive of my work since my time at NUS, long before I started my PhD.

I am very fortunate to have collaborated with Jesse Thomason and Yonatan Bisk. They taught me how to do research, pick good problems, and write papers. I have rarely met people as determined and hardworking as them, who can get things done before deadlines. They are also some of the kindest people I have met in academia and have been very supportive of my career. Another frequent collaborator I am greatly thankful to is Lucas Manuelli. Lucas taught me how to do good science by designing the right experiments and tasks. Some of my favorite projects were with Lucas like CLIPort and PERACT.

I am thankful to Chris Paxton, who taught me a lot about robots, Python, and training

neural networks. And Daniel Gordon for helping me out on my first project. Without their assistance, I would not have been able to complete several of my works.

My PhD would not have been fun without my RSE lab mates Aaron Walsman, Adam Fishman, Arun Byravan, Chris Xie, Daniel Gordon, Helen Wang, Jiafei Duan, Junha Roh, Karthik Desingh, Marius Memmel, Markus Grotz, Vinitha Ranganeni, Wentao Yuan, Xiangyun Meng, Yi Li, and Zoey Chen. Thanks for the countless trips to Don't Yell Me for bubble tea, Chillis for lunch, random PyTorch quizzes, board game nights, and lab retreats.

Outside RSE, life and work would have been dull without Aditya Kusupati, Akari Asai, Anqi Li, Aravind Rajeswaran, Boling Yang, Brian Hou, Ewin Tang, Gabor Szabo, Ivan Etimov, Jolin Xia, Jiayi, Judit Acs, Jungo Kasai, Karen Qin, Kay Ke, Matthew Schmittle, Mohak Bhardwaj, Motoyo Ohnishi, Nick Walker, Ofir Press, Patrick Lancaster, Pratyush Patel, Rosario Scalise, Sally Dong, Sasha Lambert, Sewon Min, Tim Dettmers, Victor Reis, Victor Zhong, and Yizhong Wang. Thanks for the summer hikes, road trips, bike trips, collective COVID panic, birthday lunches, dinner parties, and random conversations during robolunch.

Before my PhD, I was also part of David Hsu and Wee Sun Lee's labs at NUS. Eventhough I was an undergrad, I felt like a grad student and learnt a lot about research from Aseem Saxena, Dixant Mittal, Juekun Li, Min Chen, Neha Garg, Panpan Cai, Peter Karkus, Wei Gao, Xiao Ma, Yuanfu Luo, Zhutian Yang, and Ziquan Lan. Thanks for the afternoon foosball, snack breaks, and lunchtime discussions in tropical heat.

I could not have completed my PhD on time without the support of the amazing UW staff, including Elise deGoede Dorough, Joe Eckert, Elle Brown, Selest Nashef, Nam Pho, Stephen Spencer, and Aaron Timss. Thanks for patiently replying to all my panic emails about course registrations, CPT extensions, broken servers, and lab equipment.

I also feel fortunate to have collaborated with amazing folks from industry. My first project was a collaboration with Winson Han, Roozbeh Mottaghi, and Eric Kolve from AI2. Then I had a wonderful summer internship at MSR with Matthew Hausknecht, Eric Yuan, Marc-Alexandre Côté, and Adam Trischler, despite never stepping foot on the Microsoft campus. I also had a great time at the NVIDIA Seattle Robotics lab with Aditya Murali, Ankit Goyal, Ankur Handa, Arsalan Mousavian, Bala Sundaralingam, Caelan Garrett, Clemens Eppner, Fabio Ramos, Jonathan Tremblay, Nathan Ratliff, Kaichun Mo, Yash Narang, and Valts Blukis.

My PhD was generously supported by the Paul G. Allen Fellowship and the NVIDIA Graduate Fellowship. I would also like to extend my gratitude to UW HPC and NVIDIA for providing access to the HYAK and NGC compute clusters, respectively. And thanks to ChatGPT for fixing LaTeX tables and providing feedback on how to improve flow, structure, and grammar.

Finally, a big thank you to Ekta for all her love and support throughtout the years. Without her encouragement, I would have probably taken much longer to graduate. Moreover, she taught me a lot about architecture and how cities are built bottom-up, which later inspired me to look into bottom-up approaches for robotics. Without her delightful distractions, I might have lost sight of the world outside research.

To my family

Contents

List of Figures

List of Tables

Chapter 1

Introduction

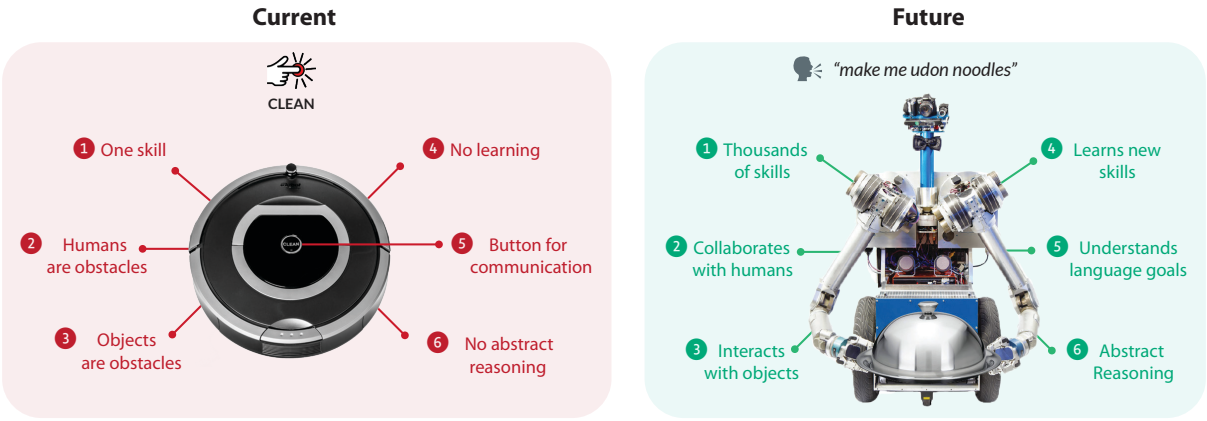


FIGURE 1.1: **Desiderata for Personal Robots.** Capabilities of a currently available personal-robot versus the desired capabilities of a general-purpose robot assistant of the future.

Language provides a natural medium for communicating goals and abstracting the physical world. The fundamental cognitive process that **connects language to perception and action** is so seamless in humans that we hardly consider it a difficult or intelligent task. As articulated in Moravec's paradox [197], machines excel at tasks that humans consider difficult, like playing Go [254] or writing a graduate-level essay [210], but even the most advanced robots struggle with tasks that require basic perception and sensorimotor skills, such as *"folding a shirt"* or *"cooking udon noodles."*

Consider the most successful personal robot — the Roomba vacuum cleaner shown in Figure 1.1. Its sole function is to clean floors, which can be activated and deactivated with the `CLEAN` button. The Roomba is a successful product precisely because it does not interact with its environment, except for sucking dust. It treats humans and objects as obstacles to be avoided, not agents to collaborate with or things to manipulate. The Roomba does not have the perception and motor skills to pick a shirt off the floor, and in fact, there is no commercially available robot that can robustly manipulate arbitrary objects in novel environments.

Moreover, it is hard to ship a Roomba without the `CLEAN` button since it is nearly impossible to predict *when* the user wants the floors to be cleaned. It could be that the user's guests are arriving early, or that the flooring is to be replaced the next day; understanding user intentions is difficult without explicit communication. Now imagine a futuristic version of the

FIGURE 1.2: **Representational Challenges in Manipulation.** Conventional representations used in robotics such as object poses, categories, instance segmentations, pre-trained vision features, and symbolic states struggle to represent states of various tasks such as (a) stretching dough, (b) mixing flour with water, (c) boiling noodles in steaming water, (d) folding dough, (e) garnishing dishes with toppings, and (f) throwing sliced dough into boiling water. *Credit: Tasty (YouTube)*

Roomba with thousands of skills, such as cooking, cleaning, and rearranging objects. With such a **general-purpose assistant** capable of multiple tasks, the issue of goal specification becomes even more critical. Adding thousands of buttons for thousands of tasks is infeasible. One natural solution for specifying goals is to use natural language. This solution has been widely successful with recent generative models like GPT-4 [210] and Stable Diffusion [232] where language has become the primary interface for guiding text [211], image [225, 232], video [109], audio [34], and 3D model [221] generation. Can we achieve similar success with language guided robotics?

Using language to guide robots has long been of interest in human-robot interaction [269]. As early as the 1960s, systems such as SHRDLU [287] were capable of manipulating simulated block shapes by following restricted language instructions. Subsequent works [271, 32, 186, 195, 277, 28, 218] have approached this problem as a *symbol grounding* issue [100], where the core challenge is mapping the syntactic structure of language to symbolic entities, such as object categories, instances, and poses. These symbolic entities are then used with high-level action operators that are executed with a separate controller. However, it is unclear if this is a universal problem formulation for robotics. Several works in cognitive psychology [89, 22], embodied cognition [11, 53], vision [147], and even robotics [35] have argued that separating perception and action in a top-down manner could be quite limiting. Consider the process of preparing hand-pulled noodles in Figure 1.2. How can the dough, flour, water mixture, steam, and other non-rigid objects be represented with conventional representations like segmentations, objects categories, poses, or symbolic predicates? It is challenging for humans to even define boundaries around all objects, because ***objects* are specific to tasks**. A strand of noodle or a particle of flour is only relevant if they need to be singled out, but otherwise, noodles and flour can be lumped together. For animals, perception and action have been interlinked since the Cambrian explosion 550 million years ago [212]. As Gibson aptly described this top-down and bottom-up process: *"we perceive in order to move, we move in order to perceive."*

One promising approach for connecting perception to action directly is through end-to-end deep learning [158]. Recent deep learning methods have successfully outperformed traditional methods from computer vision [151, 71, 224], natural language processing [69, 37], and speech processing [302], by learning intermediate representations that are directly optimized for the end task instead of using human-crafted representations. Similar to how defining what visual attributes make a chair is hard, but *recognizing a chair* is easy, it might be easier to *show how to fold a shirt*, instead of defining the physical mechanics of folding. Following this paradigm of pattern recognition, several end-to-end robot learning methods directly map camera observations to robot actions by learning from data [161, 79, 140, 174].

However, unlike vision and language domains where large amounts of data can be easily scraped from the internet, robot data is limited, expensive, and slow to collect. Most robot-learning systems are single-task agents, and even learning a robust grasping policy for picking objects could take months of data collection [141, 129]. One solution to improve data efficiency is to utilize spatial priors and symmetries in the data [313, 311]. By aligning observation and action spaces, and directly *detecting actions* in the observation space, visual affordance models are able to learn robust perception-action loops with limited data. With these spatial priors, we can focus on collecting tens of demonstrations for thousands of tasks, instead of collecting thousands of demonstrations for tens of tasks. We know from the success of vision and language models that the **diversity of tasks and datasets is crucial for generalization** in large-scale deep learning [238].

But while scaling-up simple perception-action loops is promising, it is limited to short-horizon tasks that can be completed within a few seconds. To achieve tasks that take several minutes or even hours to complete, some form of abstraction is necessary to distill the vast amount of visual and physical information. Here, natural language again provides a natural abstraction for compressing sensorimotor experiences [21, 72]. For instance, as the robot plans to gather ingredients to *"make udon noodles"*, it is inefficient and unnecessary to reason about the exact footsteps to navigate to the cabinet or anticipate the exact texture of cabinet doors to be opened. Instead, learning abstract plans such as *"go to the top cabinet"* and *"grab the flour"* are concise, generalizable, and less prone to overfitting on training environments.

In this thesis, we aim to consolidate these insights from vision, language, and robotics. This thesis does not propose novel algorithms or theoretically-motivated solutions, but proposes better problem formulations for building more capable robots. To this end, we present five key principles:

(1) Guiding robot behavior with language goals.

(2) End-to-end learning for mapping language goals and perceptual input to robot actions.

(3) Keeping the perception and action spaces aligned for data efficiency.

(4) Scaling up multi-task learning with large and diverse robot datasets.

(5) Using language to abstract complex visual and physical events.

While these principles may seem obvious, very few prior works have combined them into a unified framework. Drawing inspiration from the success of simple objectives like next token prediction in language [37], and contrastive learning in vision [47, 105], we aim to develop *generalist robots* by **learning perceptual representations of actions conditioned on language**.

The contributions of this thesis are as follows:

- In Chapter 2, we present ALFRED, a large-scale benchmark for studying agents that follow language instructions in interactive embodied environments. The benchmark presents challenging long-horizon tasks in partially-observable scenes.
- In Chapter 3, we propose CLIPort, an end-to-end manipulation agent that learns a wide variety of skills, such as folding, sweeping, and packing. Without using any symbols, segmentations, object categories or other intermediate representations, CLIPort directly encodes language goals and RGB-D images, and outputs 2D actions.
- In Chapter 4, we develop PERACT, an end-to-end Transformer the encodes language goals and RGB-D voxels, and outputs 3D actions. Like vision and language models, we show that PERACT scales to a wide variety of 6-DoF tasks and transfers to new tasks.
- In Chapter 5, we present ALFWorld, an interactive "textual" environment for learning high-level plans without complex visual and physical interactions. We show that these plans transfer well to embodied tasks.
- In Chapter 6, we consider some counterpoints to the key arguments made in this thesis. We discuss scenarios in which the proposed principles could limit the generalization capabilities of agents.
- Finally, in Chapter 7, we conclude with some takeaways and potential future directions for large-scale robot learning.

Chapter 2

Guiding Agents with Language

Language provides a natural interface for directing agents to achieve desired goals. But unlike vision-language models that ground language in just perceptual input, embodied agents need to ground language in both perception and action. A first step towards building agents that follow instructions in rich environments is to develop a standardized benchmark that evaluates grounding abilities in controlled settings.

FIGURE 2.1: **ALFRED Challenge.** ALFRED consists of 25K language directives corresponding to expert demonstrations of household tasks. We highlight several frames corresponding to portions of the accompanying language instruction. ALFRED involves interactions with objects, state changes, and long-horizon planning.

We present **ALFRED** (**A**ction **L**earning **F**rom **R**ealistic **E**nvironments and **D**irectives), a dataset and benchmark for learning a mapping from natural language instructions and egocentric vision to sequences of actions for household tasks. ALFRED includes long, compositional tasks with non-reversible state changes to shrink the gap between research benchmarks and real-world applications. ALFRED consists of expert demonstrations in interactive visual environments for 25K natural language directives. These directives contain both high-level goals like *"rinse off a mug and place it in the coffee maker"* and low-level language instructions like *"walk to the coffee maker on the right"*. ALFRED tasks are more complex in terms of sequence length, action space, and language than existing vision-and-language task datasets. The benchmark has been widely adopted by the community, and after 55 submissions across 3 years, success rates in unseen rooms have improved from 0% to 46%.

This chapter was previously published as Shridhar et al. [252]. Jesse Thomason, Daniel Gordon, Yonatan Bisk, and Winson Han contributed to the materials presented here. The code, dataset, leaderboard, and pre-trained models are available at: askforalfred.com

2.1 Overview

A robot operating in human spaces must learn to connect natural language to the physical world. This grounding problem has largely focused on connecting language to static images. However, robots need to understand task-oriented language, for example *"rinse off a mug and place it in the coffee maker"*, as illustrated in Figure 2.1.

Platforms for translating language to action have become increasingly popular, spawning new test-beds [13, 42, 44, 223]. These benchmarks include language-driven navigation and embodied question answering, which have seen dramatic improvements in modeling thanks to environments like Matterport 3D [13, 40], AI2-THOR [148], and AI Habitat [240]. However, these datasets ignore complexities arising from describing task-oriented behaviors with objects.

We introduce ALFRED, a new benchmark for connecting human language to *actions*, *behaviors*, and *objects* in interactive visual environments. Planner-based expert demonstrations are accompanied by both high- and low-level human language instructions in 120 indoor scenes in AI2-THOR 2.0 [148]. These demonstrations involve partial observability, long action horizons, underspecified natural language, and irreversible actions.

ALFRED includes 25,743 English language directives describing 8,055 expert demonstrations averaging 50 steps each, resulting in 428,322 image-action pairs. Motivated by work in robotics on segmentation-based grasping [198], agents in ALFRED interact with objects visually, specifying a pixelwise interaction mask of the target object. This inference is more realistic than simple object class prediction, where localization is treated as a solved problem. Existing beam-search [84, 265, 284] and backtracking solutions [144, 175] are infeasible due to the larger action and state spaces, long horizon, and inability to undo certain actions.

To establish baseline performance levels, we evaluate a sequence-to-sequence model akin to existing vision-and-language navigation tasks [175]. This baseline model is not effective on the complex tasks in ALFRED, achieving less than 5% success rates. For analysis, we also evaluate individual sub-goals. While performance is better for isolated sub-goals, the model lacks the reasoning capacity for long-horizon and compositional task planning.

In summary, ALFRED facilitates learning models that translate from language to sequences of actions and interactions in a visually and physically realistic simulation environment. This benchmark captures many challenges present in real-world settings for translating human language to robot actions for accomplishing household tasks. Models that can overcome these challenges will begin to close the gap towards real-world, language-driven robotics.

2.2 Related Work

Table 2.1 summarizes the benefits of ALFRED relative to other visual action datasets with language annotations.

	Language		Virtual Environment			Inference		
	# Human Annotations	Granularity	Visual Quality	Movable Objects	State Changes	Vis. Obs.	Navigation	Interaction
TACoS [229]	17k+	High&Low	Photos	✗	✗	–	–	–
R2R [13]; TD [44]	21k+; 9.3k+	Low	Photos	✗	✗	Ego	Graph	✗
EQA [60]	✗	High	Low	✗	✗	Ego	Discrete	✗
Matt. EQA [286]	✗	High	Photos	✗	✗	Ego	Discrete	✗
IQA [92]	✗	High	High	✗	✓	Ego	Discrete	Discrete
VirtualHome [223]	2.7k+	High&Low	High	✓	✓	3rd Person	✗	Discrete
VSP [323]	✗	High	High	✓	✓	Ego	✗	Discrete
ALFRED	25k+	High&Low	High	✓	✓	Ego	Discrete	Discrete + Mask

TABLE 2.1: **Dataset Comparison.** ALFRED is the first interactive visual dataset to include high-level goal and low-level natural language instructions for object and environment interactions. TACoS [229] provides detailed high- and low-level text descriptions of cooking videos, but does not facilitate task execution. For navigation, ALFRED enables discretized, grid-based movement, while other datasets use topological graph navigation or avoid navigation altogether. ALFRED requires an agent to generate spatially located interaction masks for action commands. By contrast, other datasets only require choosing from a discrete set of available interactions and object classes or offer no interactive capability.

Vision & Language Navigation. In vision-and-language navigation tasks, either natural or templated language describes a route to a goal location through egocentric visual observations [13, 42, 43, 44, 178]. Since the proposal of R2R [13], researchers have dramatically improved the navigation performance of models [84, 144, 175, 284, 285] with techniques like progress monitoring [176], as well as introduced task variants with additional, on-route instructions [206, 275]. Much of this research is limited to static environments. By contrast, ALFRED tasks include navigation, object interactions, and state changes.

Vision & Language Task Completion. There are several existing benchmarks based on simple block worlds and fully observable scenes [28, 192]. ALFRED provides more difficult tasks in richer, visually complex scenes, and uses partially observable environments. The CHAI benchmark [191] evaluates agents performing household instructions, but uses a generic "interact" action. ALFRED has seven manipulation actions, such as pick up, turn on, and open, state changes like clean versus dirty, and variation in language and visual complexity.

Previous work in the original AI2-THOR environment investigated the task of visual semantic planning [92, 323]. Artificial language came from templates, and environment interaction was handled with discrete class predictions, for example selecting *apple* as the target object from predefined options. ALFRED features human language instructions, and object selections are carried out with class-agnostic, pixelwise interaction masks. In VirtualHome [223], programs are generated from video demonstration and natural language instructions, but inference does not involve egocentric visual and action feedback or partial observability.

There is an extensive literature on language-based instruction following in the natural language processing community. There, research has focused on mapping instructions to actions [16, 43, 179, 194, 273], but these works do not involve visual, interactive environments.

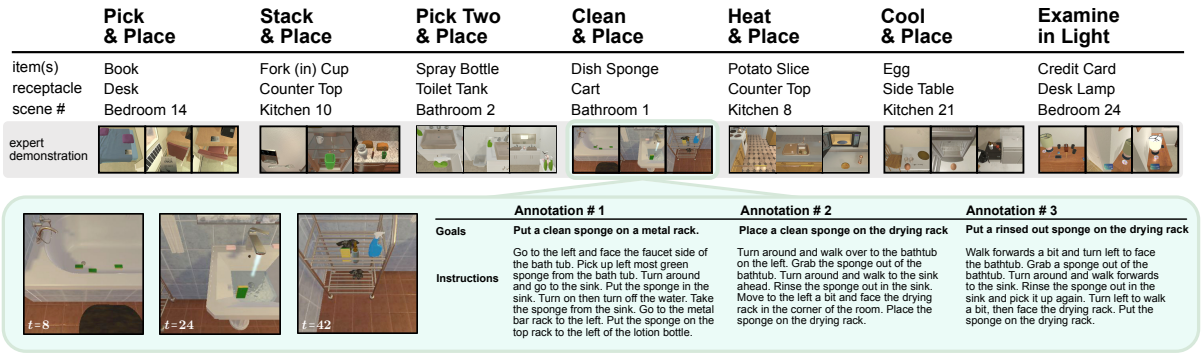


FIGURE 2.2: **ALFRED Annotations.** We introduce 7 different task types parameterized by 84 object classes in 120 scenes. An example of each task type is given above. For the **Clean & Place** demonstration, we also show the three crowdsourced language directives. Please see the supplemental material for example demonstrations and language for each task.

Embodied Question Answering. Existing datasets for visual question answering in embodied environments use templated language or static scenes [60, 92, 286, 305]. In ALFRED, rather than answering a question, the agent must complete a task specified using natural language, which requires both navigation and interaction with objects.

Instruction Alignment. Language annotations of videos enable discovering visual correspondences between words and concepts [8, 244, 229, 304, 324]. ALFRED requires performing tasks in an interactive setting as opposed to learning from recorded videos.

Robotics Instruction Following. Instruction following is a long-standing topic of interest in robotics [23, 32, 177, 193, 209, 217, 249, 277]. Lines of research consider different tasks such as cooking [32], table clearing [209], and mobile manipulation [177]. In general, they are limited to a few scenes [193], consider a small number of objects [177], or use the same environment for training and testing [23]. In contrast, ALFRED includes 120 scenes, many object classes with diverse appearances, and a test set of unseen environments.

2.3 The ALFRED Dataset

The ALFRED dataset comprises 25,743 language directives corresponding to 8,055 expert demonstration episodes. Each directive includes a high-level goal and a set of step-by-step instructions. Each expert demonstration can be deterministically replayed in the AI2-THOR 2.0 simulator.

2.3.1 Expert Demonstrations

Expert demonstrations are composed of an agent's egocentric visual observations of the environment and what action is taken at each timestep as well as ground-truth interaction masks. These demonstrations are generated by a planner [111] using metadata not available to the

agent at inference time. Navigation actions move the agent or change its camera orientation, while manipulation actions include picking and placing objects, opening and closing cabinets and drawers, and turning appliances on and off. Interactions can involve multiple objects, such as using a knife to slice an apple, cleaning a mug in the sink, and heating a potato in the microwave. Manipulation actions are accompanied by a ground truth segmentation of the target object.

Figure 2.2 gives examples of the high-level agent tasks in ALFRED, like putting a cleaned object at a destination. These tasks are parameterized by the object of focus, the destination receptacle (e.g.,, *table top*), the scene in which to carry out the task, and in the case of **Stack & Place**, a base object (e.g.,, *plate*). ALFRED contains expert demonstrations of these seven tasks executed using combinations of 58 unique object classes and 26 receptacle object classes across 120 different indoor scenes. For object classes like *potato slice*, the agent must first pick up a *knife* and find a *potato* to create slices. All object classes contain multiple visual variations with different shapes, textures, and colors. For example, there are 30 unique variants of the *apple* class. Indoor scenes include different room types: 30 each of kitchens, bathrooms, bedrooms, and living rooms.

	Train	**Validation**		**Test**	
		Seen	*Unseen*	*Seen*	*Unseen*
# Annotations	21,023	820	821	1,533	1,529
# Scenes	108	88	4	107	8

TABLE 2.2: **ALFRED Data Splits.** All expert demonstrations and associated language directives in the validation and test folds are distinct from those in the train fold. The validation and test sets are split into *seen* and *unseen* folds. Scenes in the *seen* folds of validation and test data are subsets of those in the train fold. Scenes in the *unseen* validation and test folds are distinct from the train folds and from each other.

For 2,685 combinations of task parameters, we generate three expert demonstrations per parameter set, for a total of 8,055 unique demonstrations with an average of 50 action steps. The distributions of actions steps in ALFRED demonstrations versus related datasets is given in Figure 2.3. As an example, for task parameters {task: **Heat & Place**, object: *potato*, destination: *counter top*, scene: KITCHEN-8}, we generate three different expert demonstrations by starting the agent and objects in randomly chosen locations. Object start positions have some commonsense, class-specific constraints, for example a *fork* can start inside a *drawer*, but an *apple* cannot.

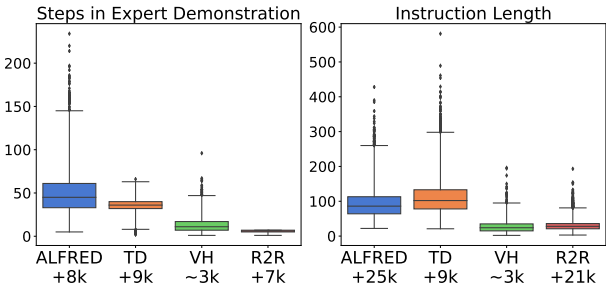


FIGURE 2.3: **Comparison to Existing Datasets.** Expert demonstration steps and instruction tokens of ALFRED compared to other datasets with human language for action sequences: Touchdown (TD) [44], Puig2018 (VH) [223], and Room-to-Room (R2R) [13].

Contrasting navigation-only datasets where expert demonstrations can come from an A^* planner, our state space includes object positions and state changes. Thus, to generate expert demonstrations we encode the agent and object states, as well as high-level environment dynamics, into Planning Domain Definition Language (PDDL) rules [88]. We then define task-specific PDDL goal conditions, for example that a heated *potato* is resting on a *table top*. Note

that the planner encodes the environment as *fully observable* and has perfect knowledge about world dynamics. For training and testing agent models, however, the environment is *partially observable*: it is only viewed through the agent's egocentric vision as actions are carried out.

We split these expert demonstrations into training, validation, and test folds (Table 2.2). Following work in vision-and-language navigation [13], we further split the validation and test into two conditions: *seen* and *unseen* environments. This split facilitates examining how well models generalize to entirely new spaces with novel object class variations.

2.3.2 Language Directives

For every expert demonstration, we collect open vocabulary, free-form language directives from at least three different annotators using Amazon Mechanical Turk (AMT), resulting in 25K total language directives. Language directives include a high-level *goal* together with low-level *instructions*, as shown in Figure 2.1 and Figure 2.2. The distribution of language annotation token lengths in ALFRED versus related datasets is given in Figure 2.3.

AMT workers are told to write instructions to tell a "smart robot" how to accomplish what is shown in a video. We create a video of each expert demonstration and segment it such that each segment corresponds to an instruction. We consult the PDDL plan for the expert demonstration to identify task sub-goals, for example the many low-level steps to navigate to a knife, or the several steps to heat a potato slice in the microwave once standing in front of it. We visually highlight action sequences related to sub-goals via colored timeline bars below the video. In each HIT (Human Intelligence Task), a worker watches the video, then writes low-level, step-by-step *instructions* for each highlighted sub-goal segment. The worker also writes a high-level *goal* that summarizes what the robot should accomplish during the expert demonstration.

These directives are validated through a second HIT by at least two annotators, with a possible third tie-breaker. For validation, we show a worker all three language directive annotations without the video. The worker selects whether the three directives describe the same actions, and if not, which is most different. If a directive is chosen as most different by a majority of validation workers, it is removed and the demonstration is subsequently re-annotated by another worker. Qualitatively, these rejected annotations contain incorrect object referents (e.g.,, *"egg"* instead of *"potato"*) or directions (e.g.,, *"go left towards..."* instead of *"right"*).

2.4 Baseline Models

An agent trained for ALFRED tasks needs to jointly reason over vision and language input and produce a sequence of low-level actions to interact with the environment.

2.4.1 Sequence-to-Sequence Models

We model the interactive agent with a CNN-LSTM sequence-to-sequence (SEQ2SEQ) architecture. A CNN enodes the visual input, a bidirectional-LSTM generates a representation of the

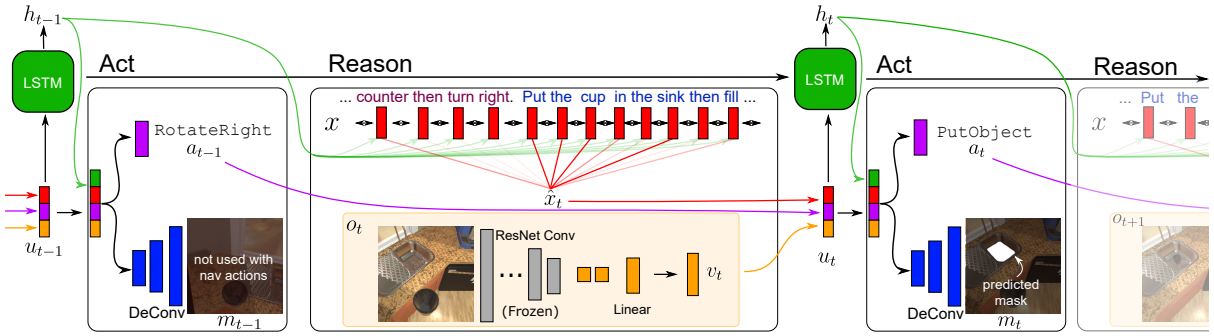


FIGURE 2.4: **Model Overview.** At each step, our model reweights the instruction based on the history ($\hat{x}_t$), and combines the current observation features (v_t) and the previously executed action (a_{t-1}). These are passed as input to an LSTM cell to produce the current hidden state. Finally, the new hidden state (h_t) is combined with the previous features to predict both the next action (a_t) and a pixelwise interaction mask over the observed image to indicate an object.

language input, and a decoder LSTM infers a sequence of low-level actions while attending over the encoded language. See Figure 2.4 for an overview and the supplementary material for implementation details.

Supervision. We train all models using imitation learning on expert trajectories. This ensures the language directives match the visual inputs. At each timestep, the model is trained to produce the expert action and associated interaction mask for manipulation actions.

We note that a DAgger-style [234] student-forcing paradigm in ALFRED is non-trivial, even disregarding language alignment. Obtaining expert demonstration actions on the fly in navigation-only datasets like R2R [13] only requires rerunning A^*. In ALFRED, on the fly demonstrations requires re-planning. In same cases re-planning is not possible: if during a task of {**Clean & Place**, *apple*, *refrigerator*, KITCHEN-3} a student-forcing model slices the only *apple* in the scene, the action cannot be recovered from and the task cannot be completed.

Visual encoding. Each visual observation o_t is encoded with a frozen ResNet-18 [107] CNN, where we take the output of the final convolution layer to preserve spatial information necessary for grounding specific objects in the visual frame. We embed this output using two more 1×1 convolution layers and a fully-connected layer. During training, a set of T observations from the expert demonstration is encoded as $\overline{V} = \langle v_1, v_2, \ldots, v_T \rangle$, where v_t is the visual feature at time-step t.

Language encoding. Given a natural language goal $\overline{G} = \langle g_1, g_2, \ldots g_{L_g} \rangle$ of L_g words, and step-by-step instructions $\overline{S} = \langle s_1, s_2 \ldots s_{L_s} \rangle$ of L_s words, we append them into a single input sequence $\overline{X} = \langle g_1, g_2, \ldots g_{L_g}, \texttt{<SEP>}, s_1, s_2 \ldots s_{L_s} \rangle$ with the <SEP> token indicating the separation between the high-level goal and low-level instructions. This sequence is fed into a bi-directional LSTM encoder to produce an encoding $x = \{x_1, \ldots, x_{L_g+L_s}\}$ for each word in $\overline{X}$.

Attention over language. The agent's action at each timestep is based on an attention mechanism weighting tokens in the instruction. We perform soft-attention on the language features x to compute the attention distribution α_t conditioned on the hidden state of the decoder h_{t-1} from the last timestep:

$$\begin{aligned} z_t &= (W_x h_{t-1})^\top x, \\ \alpha_t &= \text{Softmax}(z_t), \\ \hat{x}_t &= \alpha_t^\top x \end{aligned} \tag{2.1}$$

where W_x are learnable parameters of a fully-connected layer, z_t is a vector of scalar values that represent the attention mass for each word in x, and $\hat{x}_t$ is the weighted sum of x over the attention distribution α_t induced from z_t.

Action decoding. At each timestep t, upon receiving a new observation image o_t, the LSTM decoder takes in the visual feature v_t, language feature $\hat{x}_t$, and the previous action a_{t-1}, and outputs a new hidden state h_t:

$$\begin{aligned} u_t &= [v_t; \hat{x}_t; a_{t-1}], \\ h_t &= \text{LSTM}\ (u_t, h_{t-1}) \end{aligned} \tag{2.2}$$

where $[;]$ denotes concatenation. The hidden state h_t is used to obtain the attention weighted language feature $\hat{x}_{t+1}$.

Action and mask prediction. The agent interacts with the environment by choosing an action and producing a pixelwise binary mask indicating a specific object in the frame. Although AI2-THOR supports continuous control for agent navigation and object manipulation, we discretize the action space. The agent chooses from among 13 actions. There are 5 navigation actions: `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, and `LookDown` together with 7 interaction actions: `Pickup`, `Put`, `Open`, `Close`, `ToggleOn`, `ToggleOff`, and `Slice`. Interaction actions require a pixelwise mask to denote the object of interest.[1] Finally, the agent predicts a `Stop` action to end the episode. We concatenate the hidden state h_t with the input features u_t and train two separate networks to predict the next action a_t and interaction mask m_t:

$$\begin{aligned} a_t &= \text{argmax}\ (W_a\ [h_t; u_t])\ , \\ m_t &= \sigma\ (\textbf{deconv}\ [h_t; u_t]) \end{aligned} \tag{2.3}$$

[1]The final object chosen by the interaction API is based on the Intersection-over-Union (IoU) score between the predicted mask and the ground-truth object mask from the simulator.

where W_a are learnable parameters of a fully connected layer, **deconv** is a three-layer deconvolution network, and σ is a sigmoid activation function. Action selection is trained using softmax cross entropy with the expert action. The interaction masks are learned end-to-end in a supervised manner based on ground-truth object segmentations using binary cross-entropy loss. The mask loss is rebalanced to account for sparsity in these dense masks in which target objects can take up a small portion of the visual frame.

2.4.2 Progress Monitors

ALFRED tasks require reasoning over long sequences of images and instruction words. We propose two auxiliary losses (Eq. 2.4 & 2.5) that use additional temporal information to reduce this burden and form a sequence-to-sequence model with progress monitoring (SEQ2SEQ+PM).

Ma et al. [175] showed that agents benefit from maintaining an internal estimate of their progress towards the goal for navigation tasks. Akin to learning a value function in reinforcement learning, progress monitoring helps to learn the utility of each state in the process of achieving the overall task. Intuitively, this allows our agent to better distinguish between visually similar states such as just before putting an object in the microwave versus just after taking the object out. We introduce a simple module that predicts progress, $p_t \in [0, 1]$, conditioned on the decoder hidden state h_t and the concatenated input u_t:

$$p_t = \sigma\left(W_p\left[h_t; u_t\right]\right). \tag{2.4}$$

The supervision for p_t is based on normalized time-step values t/T, where t is the current time-step, and T is the total length of the expert demonstration (trained via L2 loss).

We also train the agent to predict the number of sub-goals completed so far, c_t. These sub-goals represent segments in the demonstration corresponding to sequences of actions like navigation, pickup, and heating as identified in the PDDL plan, discussed in Section 2.3.2. Each segment has a corresponding language instruction, but the alignment must be learned. This sub-goal prediction encourages the agent to coarsely track its progress through the language directive. This prediction is also conditioned on the decoder hidden state h_t and the concatenated input u_t:

$$c_t = \sigma\left(W_c\left[h_t; u_t\right]\right). \tag{2.5}$$

We train c_t in a supervised fashion by using the normalized number of sub-goals accomplished in the expert trajectory at each timestep, c_t/C, as the ground-truth label for a task with C sub-goals. We again train with an L2 loss.

2.5 Experiments

We evaluate the baseline models in the AI2-THOR simulator. When evaluating on test folds, we run models with the lowest validation loss. Episodes that exceed 1000 steps or cause more than 10 failed actions are terminated. Failed actions arise from bumping into walls or predicting action interaction masks for incompatible objects, such as attempting to `Pickup` a *counter*

	Validation				Test			
	Seen		*Unseen*		*Seen*		*Unseen*	
Model	Task	Goal-Cond	Task	Goal-Cond	Task	Goal-Cond	Task	Goal-Cond
NO LANGUAGE	0.0 (0.0)	5.9 (3.4)	0.0 (0.0)	6.5 (4.7)	0.2 (0.0)	5.0 (3.2)	0.2 (0.0)	6.6 (4.0)
NO VISION	0.0 (0.0)	5.7 (4.7)	0.0 (0.0)	6.8 (6.0)	0.0 (0.0)	3.9 (3.2)	0.2 (0.1)	6.6 (**4.6**)
GOAL-ONLY	0.1 (0.0)	6.5 (4.3)	0.0 (0.0)	6.8 (5.0)	0.1 (0.1)	5.0 (3.7)	0.2 (0.0)	6.9 (4.4)
INSTR-ONLY	2.3 (1.1)	9.4 (6.1)	0.0 (0.0)	**7.0** (4.9)	2.7 (1.4)	8.2 (5.5)	**0.5** (**0.2**)	7.2 (**4.6**)
SEQ2SEQ	2.4 (1.1)	9.4 (5.7)	**0.1** (0.0)	6.8 (4.7)	2.1 (1.0)	7.4 (4.7)	**0.5** (**0.2**)	7.1 (4.5)
+ PROGRESS-ONLY	2.1 (1.1)	8.7 (5.6)	0.0 (0.0)	6.9 (5.0)	3.0 (1.7)	8.0 (5.5)	0.3 (0.1)	**7.3** (4.5)
+ SUBGOAL-ONLY	2.1 (1.2)	9.6 (5.5)	0.0 (0.0)	6.6 (4.6)	3.8 (1.7)	8.9 (5.6)	**0.5** (**0.2**)	7.1 (4.5)
+ Both	**3.7** (**2.1**)	**10.0** (**7.0**)	0.0 (0.0)	6.9 (**5.1**)	**4.0** (**2.0**)	**9.4** (**6.3**)	0.4 (0.1)	7.0 (4.3)
HUMAN	-	-	-	-	-	-	91.0 (85.8)	94.5 (87.6)

TABLE 2.3: **Task and Goal-Condition Success.** For each metric, the corresponding path weighted metrics are given in parentheses. The highest values per fold and metric are shown in **blue**. All values are percentages.

top. These limitations encourage efficiency and reliability. We assess the overall and partial success of models' task executions across episodes.

2.5.1 Evaluation Metrics

ALFRED allows us to evaluate both full task and task goal-condition completion. In navigation-only tasks, one can only measure how far the agent is from the goal. In ALFRED, we can also evaluate whether task goal-conditions have been completed, for example that a *potato* has been sliced. For all of our experiments, we report both Task Success and Goal-Condition Success. Each Goal-Condition relies on multiple instructions, for example navigating to an object and then slicing it.

Task Success. Each expert demonstration is parameterized by a task to be performed, as illustrated in Figure 2.2. Task Success is defined as 1 if the object positions and state changes correspond correctly to the task goal-conditions at the end of the action sequence, and 0 otherwise. Consider the task: *"put a hot potato slice on the counter"*. The agent succeeds if, at the end of the episode, any *potato slice* object has changed to the *heated* state and is resting on any *counter top* surface.

Goal-Condition Success. The goal-condition success of a model is the ratio of goal-conditions completed at the end of an episode to those necessary to have finished a task. For example, in the previous **Heat & Place** example, there are four goal-conditions. First, a *potato* must be sliced. Second, a *potato slice* should become *heated*. Third, a *potato slice* should come to rest on a *counter top*. Fourth, the same *potato slice* that is *heated* should be on the *counter top*. If the agent slices a *potato*, then moves a slice to the counter top without heating it, then the goal-condition success score is $2/4 = 50\%$. On average, tasks in ALFRED have 2.55 goal conditions. The final score is calculated as the average goal-condition success of each episode. Task success is 1 only if goal-condition success is 1.

Path Weighted Metrics. We include a Path Weighted version of both metrics that considers the length of the expert demonstration [12]. Expert demonstrations found via a PDDL solver on global information are not guaranteed to be optimal. However, they avoid exploration, use shortest path navigation, and are generally efficient. The path weighted score p_s for metric s is given as

$$p_s = s \times \frac{L^*}{max(L^*, \hat{L})} \tag{2.6}$$

where $\hat{L}$ is the number of actions the model took in the episode, and L^* is the number of actions in the expert demonstration. Intuitively, a model receives half-credit for taking twice as long as the expert to accomplish a task.

2.5.2 Sub-Goal Evaluation

Completing the entire sequence of actions required to finish a task is challenging. In addition to assessing full task success, we study the ability of a model to accomplish the next sub-goal conditioned on the preceding expert sequence. The agent is tested by first forcing it to follow the expert demonstration to maintain a history of states leading up to the sub-goal, then requiring it to complete the sub-goal conditioned on the entire language directive and current visual observation. For the task *"put a hot potato slice on the counter"* for example, we can evaluate the sub-goal of navigating to the potato after using the expert demonstration to navigate to and pick up a *knife*. The tasks in ALFRED contain on average 7.5 such sub-goals (results in Table 2.4).

2.6 Analysis

Results from our experiments are presented in Table 2.3. We find that the initial model, without spatial or semantic maps, object segmentations, or explicit object-state tracking, performs poorly on ALFRED's long-horizon tasks with high-dimensional state-spaces. The SEQ2SEQ model achieves ∼8% goal-condition success rate, showing that the agent does learn to partially complete some tasks. This headroom (as compared with humans) motivates further research into models that can perform the complex vision-and-language planning introduced by ALFRED. The performance starkly contrasts other vision-and-language datasets focused on navigation, where sequence-to-sequence with progress monitoring performs well [175].

2.6.1 Random Agent

A random agent is commonly employed as a baseline in vision-and-language tasks. In ALFRED, an agent that chooses a uniform random action and generates a uniform random interaction mask at each timestep achieves 0% on all folds, even without an API failure limit.

2.6.2 Unimodal Ablations

Previous work established that learned agents without visual inputs, language inputs, or both performed better than random agents and were competitive with initial baselines for several navigation and question answering tasks [274]. These performance gaps were due to structural biases in the datasets or issues with model capacity. We evaluate these ablation baselines (NO LANGUAGE and NO VISION) to study vision and language bias in ALFRED.

The unimodal ablation performances in Table 2.3 indicate that both vision and language modalities are necessary to accomplish the tasks in ALFRED. The NO LANGUAGE model finishes some goal-conditions by interacting with familiar objects seen during training. The NO VISION model similarly finishes some goal-conditions by following low-level language instructions for navigation and memorizing interaction masks for common objects like *microwaves* that are centered in the visual frame.

2.6.3 Model Ablations

We additionally ablate the amount of language supervision available to the model, as language directives are given as both a high-level goal and step-by-step instructions. Providing only high-level, underspecified goal language (GOAL-ONLY) is insufficient to complete the tasks, but is enough to complete some goal-conditions. Using just low-level, step-by-step instructions (INSTR-ONLY performs similarly to using both high- and low-levels. Thus, this simple model does not seem to exploit the goal instruction to plan out sub-goals for step-by-step execution.

The two progress monitoring signals are marginally helpful, increasing the success rate from $\sim$1% to $\sim$2%. Progress monitoring leads to more efficient task completion, as indicated by the consistently higher path weighted scores. They may help avoid action repetition and with the prediction of the `Stop` action.

The agent takes more steps than the expert in all cases, as indicated by the lower path weighted scores. Sometimes, this is caused by failing to keep track of state-changes, for example heating up an *egg* in the *microwave* multiple times. Further, the models also do not generalize well to *unseen* scenes, due to the overall visual complexity in ALFRED arising from new scenes and novel object class instances.

2.6.4 Human evaluation

We obtained a human evaluation of 100 randomly sampled directives from the *unseen* test fold. The experiment involved 5 participants who completed 20 tasks each using a keyboard-and-mouse interface. Before the experiment, the participants were allowed to familiarize themselves with AI2-THOR. The action-space and task restrictions were identical to that of the baseline models. Overall, the participants obtained a high success rate of 91%, while taking slightly longer than the expert with 86% path-length weighted success rate. This indicates that the directives in ALFRED are well-aligned with the demonstrations.

		Sub-Goal Ablations - **Validation**								
	Model	Goto	Pickup	Put	Cool	Heat	Clean	Slice	Toggle	Avg.
Seen	No Lang	28	22	71	**89**	**87**	64	19	90	59
	S2S	49	32	80	87	85	**82**	23	97	67
	S2S + PM	**51**	**32**	**81**	88	85	81	**25**	**100**	**68**
Unseen	No Lang	17	9	31	75	86	13	8	4	30
	S2S	21	20	**51**	**94**	88	21	**14**	**54**	45
	S2S + PM	**22**	**21**	46	92	**89**	**57**	12	32	**46**

TABLE 2.4: **Evaluations by path weighted sub-goal success.** All values are percentages. The highest values per fold and task are shown in **blue**. We note that the NO VISION model achieves less than 2% on all sub-goals. See supplemental material for more.

2.6.5 Sub-Goal Performance

We also examine performance of the SEQ2SEQ model on individual sub-goals in ALFRED. For this experiment, we use the expert trajectory to move the agent through the episode up to the sub-task. Then, the agent begins inference based on the language directive and current visual frame.

Results from our experiments are presented in Table 2.3. We find that the initial model, without spatial or semantic maps, object segmentations, or explicit object-state tracking, performs poorly on ALFRED's long-horizon tasks with high-dimensional state-spaces. The SEQ2SEQ model achieves ∼8% goal-condition success rate, showing that the agent does learn to partially complete some tasks. This headroom (as compared with humans) motivates further research into models that can perform the complex vision-and-language planning introduced by ALFRED. The performance starkly contrasts other vision-and-language datasets focused on navigation, where sequence-to-sequence with progress monitoring performs well [175].

2.6.6 Random Agent

A random agent is commonly employed as a baseline in vision-and-language tasks. In ALFRED, an agent that chooses a uniform random action and generates a uniform random interaction mask at each timestep achieves 0% on all folds, even without an API failure limit.

2.6.7 Unimodal Ablations

Previous work established that learned agents without visual inputs, language inputs, or both performed better than random agents and were competitive with initial baselines for several navigation and question answering tasks [274]. These performance gaps were due to structural biases in the datasets or issues with model capacity. We evaluate these ablation baselines (NO LANGUAGE and NO VISION) to study vision and language bias in ALFRED.

The unimodal ablation performances in Table 2.3 indicate that both vision and language modalities are necessary to accomplish the tasks in ALFRED. The NO LANGUAGE model finishes some goal-conditions by interacting with familiar objects seen during training. The

NO VISION model similarly finishes some goal-conditions by following low-level language instructions for navigation and memorizing interaction masks for common objects like *microwaves* that are centered in the visual frame.

2.6.8 Model Ablations

We additionally ablate the amount of language supervision available to the model, as language directives are given as both a high-level goal and step-by-step instructions. Providing only high-level, underspecified goal language (GOAL-ONLY) is insufficient to complete the tasks, but is enough to complete some goal-conditions. Using just low-level, step-by-step instructions (INSTR-ONLY performs similarly to using both high- and low-levels. Thus, this simple model does not seem to exploit the goal instruction to plan out sub-goals for step-by-step execution.

The two progress monitoring signals are marginally helpful, increasing the success rate from ∼1% to ∼2%. Progress monitoring leads to more efficient task completion, as indicated by the consistently higher path weighted scores. They may help avoid action repetition and with the prediction of the `Stop` action.

The agent takes more steps than the expert in all cases, as indicated by the lower path weighted scores. Sometimes, this is caused by failing to keep track of state-changes, for example heating up an *egg* in the *microwave* multiple times. Further, the models also do not generalize well to *unseen* scenes, due to the overall visual complexity in ALFRED arising from new scenes and novel object class instances.

2.6.9 Human evaluation

We obtained a human evaluation of 100 randomly sampled directives from the *unseen* test fold. The experiment involved 5 participants who completed 20 tasks each using a keyboard-and-mouse interface. Before the experiment, the participants were allowed to familiarize themselves with AI2-THOR. The action-space and task restrictions were identical to that of the baseline models. Overall, the participants obtained a high success rate of 91%, while taking slightly longer than the expert with 86% path-length weighted success rate. This indicates that the directives in ALFRED are well-aligned with the demonstrations.

2.7 Benchmark Progress

Since the introduction of the leaderboard in 2020, the ALFRED benchmark has received 55 submissions. These submissions have come from diverse communities, including computer vision, natural language processing, robotics, and machine learning. Initially, our SEQ2SEQ models [252] had zero performance in unseen rooms and strongly overfit to training environments. Then, MOCA [257] achieved non-zero performance by factorizing perception and action into separate modules. In parallel, Episodic Transformer [214] scaled up end-to-end learning with additional synthetic data, which resulted in dramatic improvements in seen rooms but poor generalization to unseen rooms. In 2021, there was a significant jump in unseen performance

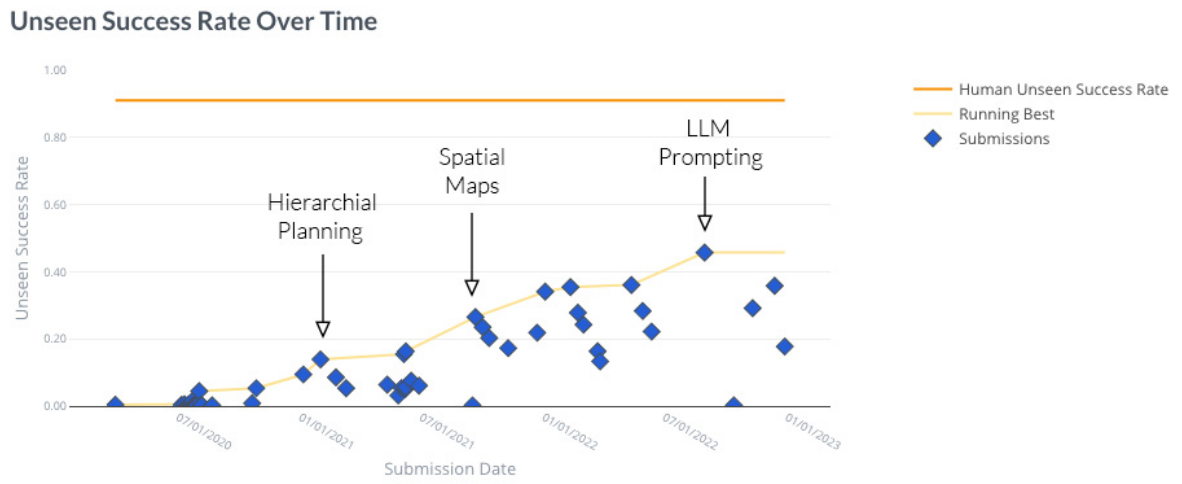


FIGURE 2.5: **Leaderboard Progress.** Success rates of agents in unseen rooms since the introduction of the ALFRED leaderboard in 2020. Our initial baselines [252] had zero-performance and struggled to achieve anything meaningful at all when put in novel environments. After 3 years and 55 submissions, the performing agent achieved an unseen success rate of 46%. This is still far from the human baseline of 91% success.

with the introduction of hierarchical planning – HiTUT [317] broke tasks in to high-level subgoals and low-level execution. The same year, HLSM [31] and FiLM [190] built 3D voxel maps of rooms for persistent spatial representations. Explicit spatial maps were found to be more effective than end-to-end approaches that relied on past observations (Transformers) or memory (LSTM/RNN) to implicitly build spatio-temporal persistence. Most recently, the hierarchical planning component was replaced with Large Language Models (LLMs) in Prompter [117], where subgoal generation was taught with a few examples without any additional fine-tuning or training.

These approaches have significantly pushed the performance of agents in both seen and unseen environments. As of 2023, the best performing agent by Inoue et al. [117] achieved a seen success rate of 53.2% and an unseen success rate of 45.6%. Surprisingly, end-to-end approaches that directly map images to actions have struggled to generalize effectively to unseen environments, primarily because of the long-horizon nature of ALFRED tasks. In fact, most high-performing agents resort to traditional robotics techniques like hierarchical planning and building spatial-maps. Nonetheless, the rapid progress has been exciting and insightful for better understanding the problems in embodied instruction-following.

2.8 Limitations

While ALFRED provides a standardized platform for evaluating instruction-following agents, its design and setup is not without limitations.

High-Level Action API. Agents interact through APIs like `MoveAhead` and `PickUp(apple)`, where the physical execution of these skills is handled by the simulator. For real-world agents like robots, it is nontrivial to acquire and execute such skills since real environments are physically complex and non-deterministic, unlike the AI2-THOR simulator. It might be possible to integrate ALFRED agents with the latest manipulation frameworks from robotics [141, 129, 251], but it may be that solving low-level control is part of the bigger problem of building general embodied agents that has to be addressed holistically.

RGB-only Restriction. Following prior benchmarks like Room-to-Room [13], ALFRED only allows RGB observations as input during evaluation. This RGB-only restriction might be slightly artificial in that real robots usually have access to other sensors like depth cameras, LiDAR, compass, and odometry. In fact, all top-ranking agents[2] in ALFRED train a separate depth-estimator with supervised learning to help build persistent spatial representations of the scene like 3D voxel maps.

Sim-to-Real Transfer. While AI2-THOR provides a rich environment for benchmarking agents, the visual and physical diversity of scenes is still very far from real-world scenes for effective sim-to-real transfer. ALFRED agents assume a deterministic and discrete action-space, and the visual fidelity of AI2-THOR might be insufficient to faithfully emulate noisy camera images. But given high-performing agents in simulation, it might be possible to replicate their performance by collecting equivalent amounts of real-world data.

Task Construction. The data generation process described in Section 2.3.1 samples tasks from a fixed distribution task categories, objects, and receptacles. This somewhat limits the diversity of tasks in the dataset, and also imposes an implicit structure on the language annotations [65]. But given that the top-ranking submission (46%) is still very far from human performance (91%), this artificial task distribution might still lead to interesting research directions and solutions.

Goodhart's Law and Human Performance. As with any benchmark in machine learning, ALFRED only captures a small fraction of the embodied experience, and is susceptible to Goodhart's Law, i.e. when metrics used to evaluate agents become the target, they seize to be good metrics. Even achieving super-human performance on the benchmark might not directly translate to real-world performance or usefulness.

[2] https://leaderboard.allenai.org/alfred/submissions/public

2.9 Discussion

We introduced ALFRED, a benchmark for learning to map language and egocentric vision to sequences of actions. ALFRED moves us closer to the community goal of language-driven robots capable of navigation and interaction in complex indoor environments. The benchmark poses several challenges including visual semantic navigation, object detection, referring expression grounding, and action grounding.

Despite these challenges, the community has made significant progress in a span of three years. Success rates in unseen rooms increased from 0% to 46%. This progress has been largely enabled by better structural priors like building occupancy maps [31], semantic vision-models [106], and hierarchical planning [317]. While this progress is encouraging, the last-mile of embodied intelligence is still bottlenecked by two key challenges: (1) acquiring low-level skills, and (2) long-horizon reasoning. We will explore learning robust low-level skills in the next two chapters, and long-horizon planning in the last chapter.

Chapter 3

Grounding Language in Precise 2D Actions

To achieve goals in the real-world, language instructions must be ultimately grounded in the low-level actions of a robot. ALFRED abstracts this problem away with a high-level interface, such as `PickUp(apple)`, which assumes providing a mask of the `apple` along with a symbolic action like `PickUp` is sufficient for physically grasping the object. But how should the robot acquire physical skills like `PickUp`? In human children, language acquisition and motor skills develop in tandem, and some evidence suggests that these abilities even influence each other [118, 162]. However, today's robots are far behind in exhibiting such human-level manipulation skills.

Recent works in robotic manipulation have shown that end-to-end networks can learn dexterous skills that require precise spatial reasoning. However, these methods fail to generalize to new goals or transfer concepts across tasks like humans. In parallel, there has been great progress in learning generalizable semantic representations for vision and language by training on large-scale internet data. But these representations lack the spatial understanding for fine-grained manipulation.

In this chapter, we propose a solution that combines the best of both worlds: a two-stream architecture with semantic and spatial pathways for vision-based manipulation. Specifically, we present CLIPort, a language-conditioned imitation-learning agent that combines the broad semantics (*what*) of CLIP [224] with the spatial precision (*where*) of Transporter [311]. Our end-to-end framework is capable of solving a variety of language-specified tabletop tasks from packing unseen objects to folding cloths, all without any explicit representations of object poses, instance segmentations, memory, symbolic states, or syntactic structures. Experiments in simulated and real-world settings show that our proposed solution is data efficient in few-shot settings and generalizes effectively to both seen and unseen semantic concepts. We even learn one multi-task policy for 10 simulated and 9 real-world tasks that is better or comparable to single-task policies.

This chapter was previously published as Shridhar et al. [250]. Lucas Manuelli contributed to some sections presented here. The code, datasets, and pre-trained models are available at cliport.github.io.

FIGURE 3.1: **Language-Conditioned Manipulation Tasks.** CLIPort is a broad framework applicable to a wide range of language-conditioned manipulation tasks in tabletop settings. We conduct large-scale experiments in Ravens [311] on 10 simulated tasks (a-j) with 1000s of unique instances per task. See Table 3.1 for challenges pertaining to each task. CLIPort can even learn one multi-task model for all 10 tasks that achieves better or comparable performance to single-task models. Similarly, we demonstrate our approach on a Franka Panda manipulator with one multi-task model for 9 real-world tasks (k-o; only 5 shown) trained with just 179 image-action pairs.

3.1 Overview

Ask a person to "get a scoop of coffee beans" or "fold the cloth in half" and they can naturally take concepts like *scoop* or *fold* and ground them in concrete physical actions within an accuracy of a few centimeters. We humans do this intuitively, without explicit geometric or kinematic models of coffee beans or cloths. Moreover, we can generalize to a broad range of tasks and concepts from a minimal set of examples on what needs to be achieved. How can we imbue robots with this ability to efficiently ground abstract semantic concepts in precise spatial reasoning?

Recently, a number of end-to-end frameworks have been proposed for vision-based manipulation [311, 6, 140, 141]. While these methods do not use any explicit representations of object poses, instance segmentations, or symbolic states, they can only replicate demonstrations with a narrow range of variability and have no notion of the semantics underlying the tasks. Switching from packing red pens to blue pens involves collecting a new training set [311], or if using goal-conditioned policies, involves the user providing a goal-image from the scene [141, 243]. In realistic human-robot interaction settings, collecting additional demonstrations or providing goal-images is often infeasible and unscalable. A natural solution to both these problems is to condition policies with natural language. Language provides an intuitive interface for

specifying goals and also for implicitly transferring concepts across tasks. While language-grounding for manipulation has been explored in the past [249, 186, 32, 195], these pipelines are limited by object-centric representations that cannot handle granular or deformable objects and often do not reason about perception and action in an integrated manner. In parallel, there has been great progress in learning models for visual representations [47, 105] and aligning representations of vision and language [171, 49, 264] by training on large-scale internet data. However, these models lack a fine-grained understanding on how to manipulate objects, i.e. physical affordances.

To this end, we propose the first framework that combines the best of both worlds: end-to-end learning for fine-grained manipulation with the multi-goal and multi-task generalization capabilities of vision-language grounding systems. We introduce a two-stream architecture for manipulation with **semantic** and **spatial** pathways broadly inspired by (or vaguely analogous to) the two-stream hypothesis in cognitive psychology [116, 169, 68]. Specifically, we present CLIPort, a language-conditioned imitation-learning agent that integrates the semantic understanding (*what*) of CLIP [224] with the spatial precision (*where*) of Transporter [311]. Transporter has been applied to a wide range of rearragement tasks from industrial packing [311] to manipulating deformable objects [243]. The key insight of the approach is formulating tabletop manipulation as a series of pick-and-place affordance predictions, where the objective is to *detect actions* rather than *detect objects* and then learn a policy. This action-centric approach to perception [89] is data efficient and effective at circumventing the need for explicit "objectness" in learnt representations. However, Transporter is a tabula rasa system that learns all visual representations from scratch and so every new goal or task requires collecting a new set of demonstrations. To address this problem, we bake in a strong semantic prior while learning policies. We condition our **semantic** stream with visual and language-goal features from a pre-trained CLIP model [224]. Since CLIP is pre-trained to align image and language features from millions of image-caption pairs from the internet, it provides a powerful prior for grounding semantic concepts that are common across tasks like categories, parts, shapes, colors, texts, and other visual attributes, all without a top-down pipeline that requires bounding boxes or instance segmentations [171, 49, 264, 142]. This allows us to formulate tabletop rearrangement as a series of language-conditioned affordance predictions, a predominantly vision-based inference problem, and thus benefit from the strengths of data-driven paradigms like scale and generalization.

To study these benefits, we conduct large-scale experiments in the Ravens [311] framework with a simulated suction-gripper robot. We propose 10 language-conditioned tasks with 1000s of unique instances per task that require both semantic and spatial reasoning (see Figure 3.1 a-j). CLIPort is not only effective at solving these tasks, but surprisingly, it can even learn a multi-task model for all 10 tasks that achieves better or comparable performance to single-task models. Further, our evaluations indicate that our multi-task model can effectively transfer attributes like "pink block" across tasks, having never seen pink blocks or the word 'pink' in the context of the evaluation task. We also demonstrate our approach on a Franka Panda manipulator with one multi-task model for 9 real-world tasks trained with just 179 image-action pairs (see Figure 3.1 k-o).

3.2 Related Work

Vision-based Manipulation. Traditionally, perception for manipulation has centered around object detectors, segmentors, and pose estimators [106, 292, 322, 316, 67, 294]. These methods cannot handle deformable objects, granular media, or generalize to unseen objects without object-specific training data. Alternatively, dense descriptors [83, 82, 263] and keypoint representations [181, 152, 166] forgo segmentation and pose representations, but do not reason about sequential actions and struggle to represent scenes with variable numbers of objects. On the other hand, end-to-end perception-to-action models can learn precise sequential policies [311, 140, 243, 310, 260, 290], but these methods have limited understanding of semantic concepts and rely on goal-images to condition policies. In contrast, Yen-Chen et. al [299] showed that pre-training on semantic tasks like classification and segmentation helps in improving efficiency and generalization of grasping predictions.

Semantic Models. With the advent of large-scale models [281, 69, 71], a number of methods for learning joint vision and language representations have been proposed [171, 49, 264, 142, 303]. However, these methods are restricted to bounding boxes or instance segmentations, which make them inapplicable for detecting things like piles of coffee beans or squares on a chessboard. Alternatively, works in contrastive learning forgo top-down object-detection and learn continuous representations by pre-training on unlabeled data [47, 105]. Recently, CLIP [224] applied a similar approach to align vision and language representations by training on millions of image-caption pairs from the internet.

Language Grounding for Robotics. Several works have proposed systems for instructing robots with natural language [249, 186, 32, 195, 28, 277, 101, 50, 29, 219, 271]. However, these methods use disentangled pipelines for perception and action with the language primarily being used to guide the perception. As such, these pipelines lack the spatial precision necessary for tasks like folding cloths. Recently, Lynch et. al [174] proposed an end-to-end system for grounding language in continuous control, but it requires several hours of human teleoperation data for a single simulated desk setting.

Two-Stream Architectures are prevalent in action-recognition networks [256, 78, 77] and audio-recognition systems [143, 293]. In robotics, Zeng et. al [313] and Jang et. al [130] have proposed two-stream pipelines for affordance predictions of novel objects. The former requires goal-images and the latter is restricted to one-step grasps with single-category goals. In contrast, our framework provides a rich and intuitive interface with composable language commands for sequential tasks.

3.3 CLIPort

CLIPort is an imitation-learning agent based on four key principles: (1) Manipulation through a two-step primitive where each action involves a start and final end-effector pose. (2) Visual

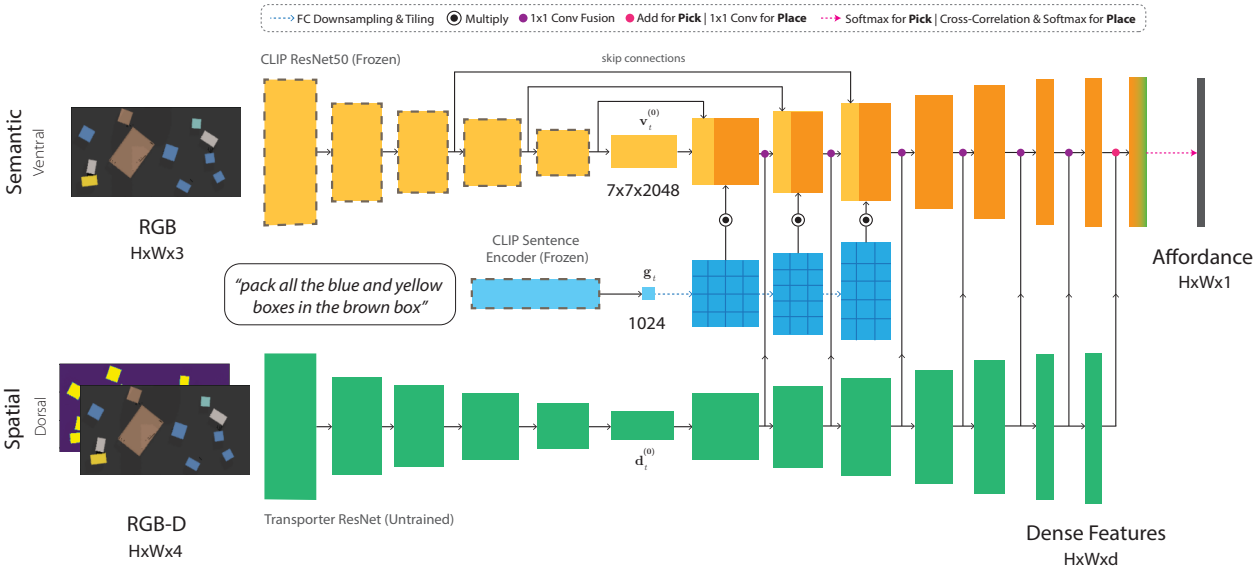


FIGURE 3.2: **CLIPort Two-Stream Architecture.** An overview of the **semantic** and **spatial** streams. The **semantic** stream uses a frozen CLIP ResNet50 [224] to encode RGB input, and its decoder layers are conditioned with tiled language features from the CLIP sentence encoder. The **spatial** stream encodes RGB-D input, and its decoder layers are laterally fused with the **semantic** stream. The final output is a map of dense pixelwise features that is used for pick or place affordance predictions. This same two-stream architecture is used in all 3 Fully-Convolutional-Networks $f_{\text{pick}}, \Phi_{\text{query}}$, and Φ_{key} with f_{pick} is used to predict pick actions, and Φ_{query} and Φ_{key} are used to predict place actions.

representations of actions that are equivariant to translations and rotations [149, 55]. (3) Two separate pathways for semantic and spatial information. (4) Language-conditioned policies for specifying goals and also transferring concepts across tasks. Combining (1) and (2) from Transporter with (3) and (4) allows us to achieve generalizable policies that go beyond just imitating demonstrations.

Section 3.3.1 describes the problem formulation, gives an overview of Transporter [311], and presents our language-conditioned model. Section 3.3.2 provides details on the training approach.

3.3.1 Language-Conditioned Manipulation

We consider the problem of learning a goal-conditioned policy π that outputs actions $\mathbf{a}_t$ given input $\gamma_t = (\mathbf{o}_t, \mathbf{l}_t)$ consisting of a visual observation $\mathbf{o}_t$ and an English language instruction $\mathbf{l}_t$:

$$\pi(\gamma_t) = \pi(\mathbf{o}_t, \mathbf{l}_t) \rightarrow \mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}) \in \mathcal{A} \tag{3.1}$$

The actions $\mathbf{a} = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}})$ specify the end-effector pose for picking and placing, respectively. We consider tabletop tasks where $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}} \in \mathbf{SE}(2)$. The visual observation $\mathbf{o}_t$ is a top-down orthographic RGB-D reconstruction of the scene where each pixel corresponds to a point in 3D space. The language instruction $\mathbf{l}_t$ either specifies step-by-step instructions e.g.

"pack the scissors" $\rightarrow$ *"pack the purple tape"* $\rightarrow$ etc., or a single goal description for the whole task e.g *"pack all the blue and yellow boxes in the brown box"*. See Figure 3.6 for specific examples.

We assume access to a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, \ldots, \zeta_n\}$ of n expert demonstrations with associated discrete-time input-action pairs $\zeta_i = \{(\mathbf{o}_1, \mathbf{l}_1, \mathbf{a}_1), (\mathbf{o}_2, \mathbf{l}_2, \mathbf{a}_2), \ldots\}$ where $\mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}})$ corresponds to expert pick-and-place coordinates at timestep t. These expert demonstrations are used to supervise the policy π.

Transporter for Pick-and-Place. The policy π is trained with Transporter [311] to perform spatial manipulation. The model first (i) attends to a local region to decide where to pick, then (ii) computes a placement location by finding the best match through cross-correlation of deep visual features.

Following Transporter [311, 243], the policy π is composed of two action-value modules (Q-functions): The pick module $\mathcal{Q}_{\text{pick}}$ decides where to pick, and conditioned on this pick action the place module $\mathcal{Q}_{\text{place}}$ decides where to place. These modules are implemented as Fully-Convolutional-Networks (FCNs) that are translationally equivariant by design. As we will describe in more detail below, we extend these networks to two-stream architectures that can handle language input. The pick FCN f_{pick} takes input $\gamma_t = (\mathbf{o}_t, \mathbf{l}_t)$ and outputs a dense pixelwise prediction $\mathcal{Q}_{\text{pick}} \in \mathbb{R}^{H \times W}$ of action-values, where are used to predict the pick action $\mathcal{T}_{\text{pick}}$:

$$\mathcal{T}_{\text{pick}} = \underset{(u,v)}{\text{argmax}}\, \mathcal{Q}_{\text{pick}}((u,v)|\gamma_t) \tag{3.2}$$

Since $\mathbf{o}_t$ is an orthographic heightmap, each pixel location (u, v) can be mapped to a 3D picking location using the known camera calibration. f_{pick} is trained in a supervised manner to predict the pick action $\mathcal{T}_{\text{pick}}$ that imitates the expert demonstration with the specified language instruction at timestep t.

The second FCN Φ_{query} takes in $\gamma_t[\mathcal{T}_{\text{pick}}]$, which is a $c \times c$ crop of $\mathbf{o}_t$ centered at $\mathcal{T}_{\text{pick}}$ along with the language instruction $\mathbf{l}_t$, and outputs a query feature embedding of shape $\mathbb{R}^{c \times c \times d}$. The third FCN Φ_{key} consumes the full input γ_t and outputs a key feature embedding of shape $\mathbb{R}^{H \times W \times d}$. The place action-values $\mathcal{Q}_{\text{place}}$ are then computed by cross-correlating the query and key features:

$$\mathcal{Q}_{\text{place}}(\Delta\tau|\gamma_t, \mathcal{T}_{\text{pick}}) = \big(\Phi_{\text{query}}(\gamma_t[\mathcal{T}_{\text{pick}}]) * \Phi_{\text{key}}(\gamma_t)\big)[\Delta\tau] \tag{3.3}$$

where $\Delta\tau \in \boldsymbol{SE}(2)$ represents a potential placement pose. Since $\mathbf{o}_t$ is an orthographic heightmap, rotations in the placement pose $\Delta\tau$ can be captured by stacking k discrete angle rotations of the crop before passing it through the query network Φ_{query}. Then for the place action $\mathcal{T}_{\text{place}} = \text{argmax}_{\Delta\tau}\, \mathcal{Q}_{\text{place}}(\Delta\tau|\gamma_t, \mathcal{T}_{\text{pick}})$, where the $\mathcal{Q}_{\text{place}}$ is trained to imitate the placements in the expert demonstrations. For all models, we use $c = 64$, $k = 36$ and $d = 3$. As in Transporter [311, 243], our framework can be extended to handle any motion primitive like pushing, sliding, etc. that can be parameterized by two end-effector poses at each timestep. For more details, we refer the reader to the original paper [311].

Two-Stream Architecture. In CLIPort, we extend the network architecture of all three FCNs $f_{\text{pick}}, \Phi_{\text{query}}$ and Φ_{key} from Transporter [311] to allow for language input and reasoning about

high-level semantic concepts. We extend the FCNs to two-pathways: **semantic** (ventral) and **spatial** (dorsal). The **semantic** stream is conditioned with language features at the bottleneck and fused with intermediate features from the **spatial** stream. See Figure 3.2 for an overview of the architecture.

The **spatial** stream is identical to the ResNet architecture in Transporter – a tabula rasa network that takes in RGB-D input $\mathbf{o}_t$ and outputs dense features through an hourglass encoder-decoder model. The **semantic** stream uses a frozen pre-trained CLIP ResNet50 [224] to encode the RGB input[1] $\tilde{\mathbf{o}}_t$ up until the penultimate layer $\tilde{\mathbf{o}}_t \rightarrow \mathbf{v}_t^{(0)} : \mathbb{R}^{7\times7\times2048}$, and then introduces decoding layers that upsample the feature tensors to mimic the **spatial** stream $\mathbf{v}_t^{(l-1)} \rightarrow \mathbf{v}_t^{(l)} : \mathbb{R}^{h\times w\times C}$ at each layer l.

The language instruction $\mathbf{l}_t$ is encoded with CLIP's Transformer-based sentence encoder to produce a goal encoding $\mathbf{l}_t \rightarrow \mathbf{g}_t : \mathbb{R}^{1024}$. This goal encoding $\mathbf{g}_t$ is downsampled with fully-connected layers to match the channel dimension C and tiled to match the spatial dimensions of the decoder features such that $\mathbf{g}_t \rightarrow \mathbf{g}_t^{(l)} : \mathbb{R}^{h\times w\times C}$. The decoder features are then conditioned with the tiled goal features through an element-wise product $\mathbf{v}_t^{(l)} \odot \mathbf{g}_t^{(l)}$ (Hadamard product). Since CLIP was trained with contrastive loss on the dot-product alignment between pooled image features and language encodings, the element-wise product allows us to use this alignment while the tiling preserves the spatial dimensions of the visual features. This language conditioning is repeated for three subsequent layers after the bottleneck inspired by LingUNet [191]. We also add skip connections to these layers from the CLIP ResNet50 encoder to utilize different levels of semantic information from shapes to parts to object-level concepts [90]. Finally, following existing two-stream architectures in video-action recognition [77], we add lateral connections from the **spatial** stream to the **semantic** stream. These connections involve concatenating two feature tensors and applying 1×1 `conv` to reduce the channel dimension $[\mathbf{v}_t^{(l)} \odot \mathbf{g}_t^{(l)}; \mathbf{d}_t^{(l)}] : \mathbb{R}^{h\times w\times C_\mathbf{v}+C_\mathbf{d}} \rightarrow \mathbb{R}^{h\times w\times C_\mathbf{v}}$, where $\mathbf{v}_t^{(l)}$ and $\mathbf{d}_t^{(l)}$ are the **semantic** and **spatial** tensors at layer l, respectively. For the final fusion of dense features, `addition` for f_{pick} and 1×1 `conv` fusion for Φ_{query} and Φ_{key} worked the best empirically. See Appendix B.4 for details on the exact architecture.

3.3.2 Implementation Details

Training from demonstrations. Similar to Transporter [311] we train CLIPort through imitation learning from a set of expert demonstrations $\mathcal{D} = \{\zeta_1, \zeta_2, \ldots, \zeta_n\}$ consisting of discrete-time input-action pairs $\zeta_i = \{(\mathbf{o}_1, \mathbf{l}_1, \mathbf{a}_1), (\mathbf{o}_2, \mathbf{l}_2, \mathbf{a}_2), \ldots\}$. During training, we randomly sample an input-action pair from the dataset and supervise the model end-to-end with one-hot pixel encodings of demonstration actions $Y_{\text{pick}} : \mathbb{R}^{H\times W\times k}$ and $Y_{\text{place}} : \mathbb{R}^{H\times W\times k}$ with k discrete rotations. In simulated experiments with the suction-gripper, we use $k = 1$ for pick actions and $k = 36$ for place actions. The model is trained with cross-entropy loss like a classifier: $\mathcal{L} = -\mathbb{E}_{Y_{\text{pick}}}[\log \mathcal{V}_{\text{pick}}] - \mathbb{E}_{Y_{\text{place}}}[\log \mathcal{V}_{\text{place}}]$ where $\mathcal{V}_{\text{pick}} = \text{softmax}(\mathcal{Q}_{\text{pick}}((u, v)|\gamma_t))$ and $\mathcal{V}_{\text{place}} = \text{softmax}(\mathcal{Q}_{\text{place}}((u', v', \omega')|\gamma_t, \mathcal{T}_{\text{pick}}))$. Compared to the original Transporter models

[1] We cannot use depth information with CLIP since it was trained with RGB-only image-caption pairs from the internet.

Task	precise placing	multimodal placing	multi-step sequencing	unseen poses	unseen colors	unseen objects	language instruction
put-blocks-in-bowls-seen-colors*	✗	✓	✗	✓	✗	✗	goal
put-blocks-in-bowls-unseen-colors*	✗	✓	✗	✓	✓	✗	goal
assembling-kits-seq-seen-colors	✓	✓	✓	✓	✗	✓	step
assembling-kits-seq-unseen-colors	✓	✓	✓	✓	✓	✓	step
packing-unseen-shapes	✗	✓	✗	✓	✓	✓	goal
stack-block-pyramid-seq-seen-colors	✓	✓	✓	✓	✗	✗	step
stack-block-pyramid-seq-unseen-colors	✓	✓	✓	✓	✓	✗	step
towers-of-hanoi-seq-seen-colors	✓	✓	✓	✓	✗	✗	step
towers-of-hanoi-seq-unseen-colors	✓	✓	✓	✓	✓	✗	step
packing-box-pairs-seen-colors*§	✓	✓	✓	✓	✗	✓	goal
packing-box-pairs-unseen-colors*§	✓	✓	✓	✓	✓	✓	goal
packing-seen-google-objects-seq§	✗	✓	✓	✓	✗	✗	step
packing-unseen-google-objects-seq§	✗	✓	✓	✓	✓	✓	step
packing-seen-google-objects-group*§	✗	✓	✗	✓	✗	✗	goal
packing-unseen-google-objects-group*§	✗	✓	✗	✓	✓	✓	goal
align-rope*†	✓	✓	✓	✓	✗	✗	goal
separating-piles-seen-colors*†	✓	✓	✓	✓	✗	✗	goal
separating-piles-unseen-colors*†	✓	✓	✓	✓	✓	✗	goal

§tasks that are commonly found in industry.
*tasks that have more than one correct sequence of actions.
†tasks that require manipulating deformable objects and granular media.

TABLE 3.1: Language-conditioned tasks in Ravens [311] with their associated challenges.

that were trained for 40K iterations, we train our models for 200K iterations (with data augmentation; see Appendix B.5) to account for additional semantic variation in tasks – randomized colors, shapes, objects. All models are trained on a single commodity GPU for 2 days with a batch size of 1.

Training multi-task models. Multi-task training is nearly identical to single-task training except for the sampling of training data. First, we randomly sample a task, and then select a random input-action pair from that task in the dataset. Using this strategy, all tasks are equally likely to be sampled but longer horizon tasks are less likely to reach full coverage of input-action pairs available in the dataset. To compensate for this, we train all multi-task models 3× longer for 600K iterations or 6 GPU days.

3.4 Results

We perform experiments both in simulation and hardware aimed at answering the following questions: 1) How effective is the language-conditioned two-stream architecture for fine-grained manipulation compared to one-stream alternatives and other simpler baselines? 2) Is it possible to train a multi-task model for all tasks, and how well does it perform and generalize? 3) How well do these models generalize to seen and unseen semantic attributes like colors, shapes, and object categories?

3.4.1 Simulation Setup

Environment. All simulated experiments are based on a Universal Robot UR5e with a suction gripper. The setup provides a systematic and reproducible environment for evaluation, especially for benchmarking the ability to ground semantic concepts like colors and object categories. The input observation is a top-down RGB-D reconstruction from 3 cameras positioned around a rectangular table: one in the front, one on the left shoulder, and one on the right shoulder, all pointing towards the center. Each camera has a resolution of 640×480 and is noiseless.

Language-Conditioned Manipulation Tasks. We extend the Ravens benchmark [311] set in PyBullet [58] with 10 language-conditioned manipulation tasks. See Figure 3.1 for examples and Table 3.1 for challenges associated with each task. Each task instance is constructed by sampling a set of objects and attributes: poses, colors, sizes, and object categories. 8 of the 10 tasks have two variants, denoted by seen and unseen, depending on whether the task has unseen attributes (e.g. color) at test time. For colors, there are 3 seen $\mathbb{T}_{\text{seen colors}} = \{\texttt{yellow, brown, gray, cyan}\}$ and 3 unseen $\mathbb{T}_{\text{unseen colors}} = \{\texttt{orange, purple, pink, white}\}$ with 3 overlapping colors $\mathbb{T}_{\text{all}} = \{\texttt{red, green, blue}\}$ used in both the seen and unseen spilts. For packing objects, we use 56 tabletop objects from the Google Scanned Objects dataset [73] and split them into 37 seen and 19 unseen objects. Figure 3.4 presents the full list of attributes, shapes, and objects across seen and unseen splits. The instructions are constructed from templates for simulated experiments, and human-annotated for real experiments. see Appendix B.1 for additional details.

Evaluation Metric. We adopt the 0 (fail) to 100 (success) scores proposed in the Ravens benchmark [311]. The score assigns partial credit based on the task, e.g. $3/5 \Rightarrow 60.0$ for packing 3 out of 5 objects specified in the instructions, or $30/56 \Rightarrow 53.6$ for pushing 30 out of 56 particles into the correct zone. See Appendix B.1 for the specific evaluation metric used in each task. During an evaluation episode, an agent keeps interacting with the scene until an oracle indicates task-completion. We report scores on 100 evaluation runs for agents trained with $n = 1, 10, 100, 1000$ demonstrations.

3.4.2 Simulation Results

Table 3.2 presents results from our large-scale experiments in Ravens and Figure 3.3 summarizes these results with average scores across seen and unseen splits.

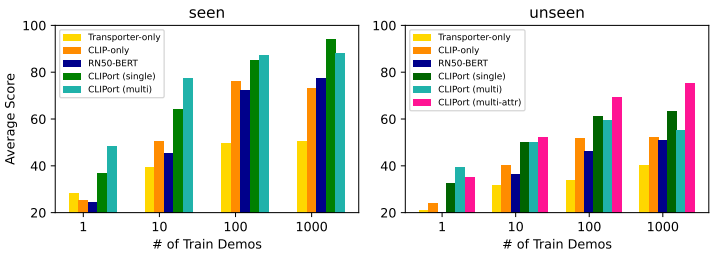


FIGURE 3.3: Average scores across seen and unseen splits for all tasks in Table 3.2.

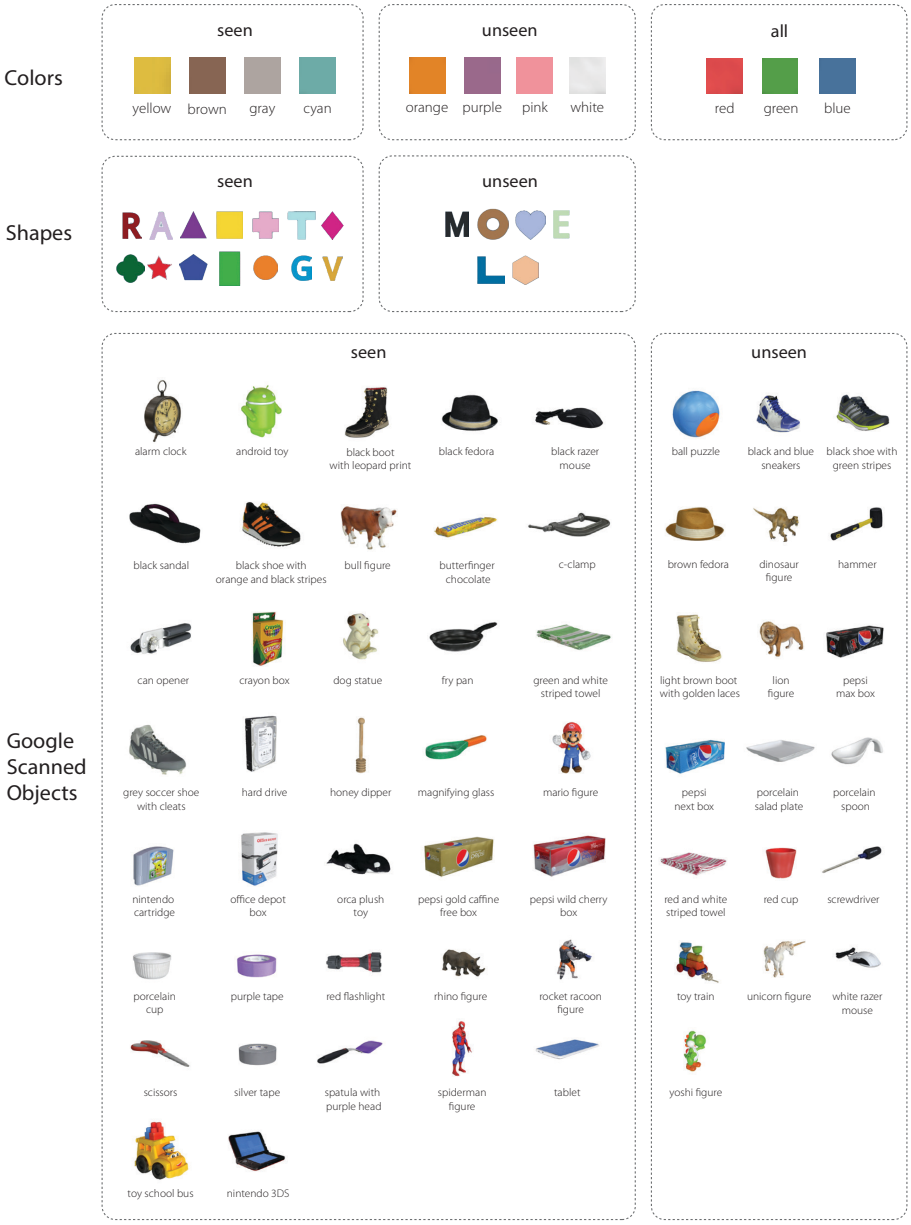


FIGURE 3.4: **Attributes and Objects.** Attributes and objects across seen and unseen splits. Shapes objects are from Transporter [311]. Other tabletop objects are from the Google Scanned Objects dataset [73]

Baseline Methods. To study the effectiveness of our two-stream architecture, we broadly compare against two baselines: Transporter-only and CLIP-only. Transporter-only is the original Transporter [311], or equivalently, the **spatial** stream of CLIPort with RGB-D input. Although Transporter-only does not receive any language goals, it shows what can be achieved through chance by exploiting the most likely actions seen during training. On the other hand, CLIP-only is just the **semantic** stream of CLIPort with RGB and language input. CLIP-only shows what can be achieved by fine-tuning a pre-trained semantic model for manipulation without spatial information, particularly depth.

Two-Stream Performance. Figure 3.3 (seen) captures the essence of our main claims. The performance of Transporter-only saturates at 50% since it doesn't use the language instruction to ground the desired goal. CLIP-only does have a goal, but lacks the spatial precision to go the last mile and thus saturates at 76%. Only CLIPort (single) achieves more than 90%, which indicates that both the semantic and spatial streams are crucial for fine-grained manipulation. Further, CLIPort (single) achieves 86% on most tasks with just 100 demonstrations, showcasing its efficiency.

In addition to these baselines, we present various ablations and alternative one-stream and two-stream models in Section 3.4.3. To briefly summarize these results, CLIP is essential for few-shot learning (i.e. $n \geq 10$) in lieu of **semantic** stream alternatives like ImageNet-trained ResNet50 [161] with BERT [69]. Image-goal models outperform CLIPort (single) in packing Google objects, but this is only because they do not have to solve the language-grounding problem.

Multi-Task Performance. In realistic scenarios, we want the robot to be capable of any task, not just one task. We investigate this through CLIPort (multi) in Table 3.2 with one multi-task model trained on all 10 tasks. CLIPort (multi) models are trained only on seen-splits of tasks, so an unseen attribute like 'pink' is consistent throughout single and multi-task settings. Surprisingly, CLIPort (multi) outperforms single-task CLIPort (single) models in $41/72 = 57\%$ of the evaluations in Table 3.2. This trend is also evident in Figure 3.3 (seen), especially in instances with 100 demonstrations or less. Although CLIPort (multi) is trained on more diverse data from other tasks, both CLIPort (multi) and CLIPort (single) have access to the same amount of data *per* task. This supports our premise that language is a strong conditioning mechanism for reusing concepts from other tasks without learning them from scratch. It also validates a trait of data-driven approaches where training on lots of diverse data leads to more robust and generalizable representations [224, 172]. However, CLIPort (multi) performs worse on longer-horizon tasks like align-rope. We hypothesize that this is because longer-horizon tasks get less coverage of input-action pairs in the dataset. Future works could use better sampling methods that balance tasks according to their average time horizon.

Generalizing to Unseen Attributes. Tasks that require generalizing to novel colors, shapes, and objects are more difficult and all our agents achieve relatively lower performance on these tasks, as shown in Figure 3.3 (unseen). However, CLIPort (single) models do substantially better than chance, i.e., Transporter-only. The lower performances are due to the difficulty of grounding unseen attributes such as 'pink' and 'orange' in the language instruction "put the pink block on the orange bowl", when the agent has never encountered words 'orange', 'pink' or their corresponding visual characteristics in the context of the physical environment. Although pre-trained CLIP has been exposed to the attribute 'pink', it could correspond to different concepts in the physical setting depending on factors like lighting condition, and thus

	packing-box-pairs seen-colors				packing-box-pairs unseen-colors				packing-seen-google objects-seq				packing-unseen-google objects-seq				packing-seen-google objects-group				packing-unseen-google objects-group			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [311]	44.2	55.2	54.2	52.4	34.6	48.7	47.2	54.1	26.2	39.7	45.4	46.3	19.9	29.8	28.7	37.3	60.0	54.3	61.5	59.9	46.2	54.7	49.8	52.0
CLIP-only	38.6	69.7	88.5	87.1	33.0	65.5	68.8	61.2	29.1	67.9	**89.3**	95.8	37.1	49.4	60.4	57.8	52.5	62.0	**89.6**	**92.7**	43.4	65.9	73.1	70.0
RN50-BERT	36.2	64.0	**94.7**	90.3	31.4	52.7	65.6	**72.1**	32.9	48.4	87.9	94.0	29.3	48.5	48.3	56.1	46.4	52.9	76.5	86.4	43.2	52.0	66.3	73.7
CLIPort (single)	51.6	82.9	92.7	**98.2**	45.6	65.3	68.6	71.5	14.8	59.5	86.8	**96.2**	27.2	50.0	65.5	**71.9**	52.7	67.0	84.1	94.0	61.5	66.2	78.4	**81.5**
CLIPort (multi)	**66.8**	**88.6**	94.1	96.6	**59.0**	**69.7**	**76.2**	71.4	**41.6**	**78.4**	85.0	84.4	**40.7**	**51.1**	**65.8**	70.3	**71.3**	**84.6**	**89.6**	88.3	**68.4**	**69.6**	**78.4**	80.3
CLIPort (multi-attr)	–	–	–	–	*46.2*	*72.0*	*86.2*	*80.3*	–	–	–	–	*35.4*	*45.1*	*78.9*	*87.4*	–	–	–	–	*48.6*	*69.3*	*84.8*	*89.1*

	stack-block-pyramid seq-seen-colors				stack-block-pyramid seq-unseen-colors				separating-piles seen-colors				separating-piles unseen-colors				towers-of-hanoi seq-seen-colors				towers-of-hanoi seq-unseen-colors			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [311]	4.5	2.3	5.2	4.5	3.0	4.0	2.3	5.8	42.7	52.3	42.0	48.4	41.2	49.2	44.7	52.3	25.4	67.9	98.0	99.9	24.3	44.6	71.7	80.7
CLIP-only	6.3	28.7	55.7	54.8	2.0	12.2	18.3	19.5	43.5	55.0	84.9	90.2	**59.9**	49.6	73.0	71.0	9.4	52.6	88.6	45.3	24.7	47.0	67.0	58.0
RN50-BERT	5.3	35.0	89.0	97.5	6.2	12.2	21.5	30.7	31.8	47.8	46.5	46.5	33.4	44.4	41.3	44.9	28.0	66.1	91.3	92.1	17.4	75.1	85.3	89.3
CLIPort (single)	28.3	64.7	93.3	**98.8**	13.7	24.3	31.2	**41.3**	**54.5**	59.5	**93.1**	**98.0**	47.2	51.0	**76.6**	**75.2**	59.4	92.9	97.4	**100**	56.1	**89.7**	**95.9**	**99.4**
CLIPort (multi)	**33.5**	**75.3**	**96.8**	96.5	**23.3**	**26.8**	**31.7**	22.2	48.9	**72.4**	90.3	89.0	56.6	**62.6**	64.9	62.8	**61.6**	**96.3**	**98.7**	98.1	**60.1**	65.6	76.7	68.7
CLIPort (multi-attr)	–	–	–	–	*15.5*	*51.5*	*59.3*	*79.8*	–	–	–	–	*49.9*	*51.8*	*48.2*	*59.8*	–	–	–	–	*56.7*	*78.0*	*88.3*	*96.9*

	align-rope				packing-unseen-shapes				assembling-kits-seq seen-colors				assembling-kits-seq unseen-colors				put-blocks-in-bowls seen-colors				put-blocks-in-bowls unseen-colors			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [311]	6.9	30.6	33.1	51.5	16.0	20.0	22.0	22.0	5.8	11.6	28.6	29.6	7.8	17.6	25.6	28.4	16.8	33.3	62.7	64.7	11.7	17.2	14.8	18.7
CLIP-only	13.4	48.7	70.4	70.7	13.0	28.0	**44.0**	**50.0**	0.8	9.2	19.8	23.0	2.0	4.6	10.8	19.8	23.5	60.2	93.5	97.7	11.2	34.2	33.2	44.5
RN50-BERT	3.1	25.0	63.8	57.1	19.0	25.0	32.0	44.0	2.2	5.6	11.6	21.8	1.6	6.4	10.4	18.4	13.8	44.5	81.2	91.8	16.2	23.0	30.3	23.8
CLIPort (single)	**20.1**	**77.4**	**85.6**	**95.4**	21.0	26.0	40.0	37.0	**12.2**	17.8	**47.0**	**66.6**	**16.2**	18.0	**35.4**	**34.8**	23.5	68.3	92.5	**100**	18.0	35.3	37.3	25.0
CLIPort (multi)	19.6	49.3	82.4	74.9	**25.0**	**35.0**	37.0	31.0	11.4	**34.8**	46.2	52.4	7.8	**21.6**	29.0	25.4	**54.0**	**90.2**	**99.5**	**100**	**32.0**	**48.8**	**55.3**	**45.8**
CLIPort (multi-attr)	–	–	–	–	–	–	–	–	–	–	–	–	*7.6*	*10.4*	*43.8*	*34.6*	–	–	–	–	*23.0*	*41.8*	*66.5*	*75.7*

TABLE 3.2: **Language-Conditioned Test Results.** Task success scores (mean %) from 100 evaluation instances vs. # of training demonstrations (1, 10, 100, or 1000). The challenges pertaining to each task are described in Appendix B.1. CLIPort (single) models are trained on seen splits, and evaluated on both seen and unseen splits. CLIPort (multi) models are trained on seen splits of all 10 tasks with 1$\mathbb{T}$, 10$\mathbb{T}$, 100$\mathbb{T}$, and 1000$\mathbb{T}$ demonstrations where $\mathbb{T} = 10$. CLIPort (multi-attr) indicate CLIPort (multi) models trained on seen-and-unseen splits from all tasks *except* for that one particular heldout task, for which it is trained only the seen split. See Figure 3.3 for an overview with average scores.

requires at least few examples to condition the trainable **semantic** decoder layers. Additionally, we notice that CLIPort (single) is also less prone to overfitting compared to Transporter-only. As evidenced in towers-of-hanoi-seq-unseen-colors task in Table 3.2, Transporter-only suffers from a performance drop because of rings with unseen colors despite the fact that Tower of Hanoi can be solved without attending to the colors and simply focusing on the ring size. We hypothesize that since CLIP was trained on diverse internet data, it enables our agent to focus on task-relevant concepts while ignoring irrelevant aspects of the task.

Transferring Attributes across Tasks. One solution for dealing with unseen attributes is to explicitly learn these attributes from other tasks. We study this with CLIPort (multi-attr) in Table 3.2 and Figure 3.3 (unseen). For these models, CLIPort (multi) is trained on both seen-and-unseen splits from all tasks *except* for the task being evaluated on, for which it was only trained on the seen split. As such, this evaluation measures whether having seen pink blocks in put-blocks-in-bowl-unseen-colors helps solve "pack all the pink and cyan boxes" in packing-box-pairs-unseen-colors. Results indicate that such explicit transfers result in significant improvements. For instance, on the put-blocks-in-bowls-unseen-colors task for $n = 1000$, CLIPort (multi)'s performance increases from 45.8 to 75.7.

3.4.3 Model Ablations

Table 3.3 presents various baselines and model ablations from our simulated experiments. The following is a description of each model:

	stack-block-pyramid seq-seen-colors				stack-block-pyramid seq-unseen-colors				packing-seen-google object-seq				packing-unseen-google object-seq			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
One-Stream Transporter-only	4.5	2.3	5.2	4.5	3.0	4.0	2.3	5.8	26.2	39.7	45.4	46.3	19.9	29.8	28.7	37.3
One-Stream CLIP-only	6.3	28.7	55.7	54.8	2.0	12.2	18.3	19.5	52.5	62.0	**89.6**	92.7	43.4	**65.9**	**73.1**	70.0
One-Stream Language Tptr	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.1	0.0	0.0
One-Stream Image-Goal Tptr	1.8	1.3	7.0	6.8	2.5	4.7	4.2	4.8	**64.5**	**67.0**	81.8	85.4	**47.7**	62.8	71.0	**83.3**
Two-Stream CLIP-Tptr w/o skip	0.0	4.3	3.8	3.3	4.2	5.2	3.2	2.5	22.9	26.1	36.9	38.9	24.4	29.9	33.7	38.3
Two-Stream Untrained-Sem-Tptr	3.0	12.7	61.5	51.2	1.0	6.8	17.2	15.7	28.8	40.5	67.1	79.7	27.2	34.7	33.0	34.8
Two-Stream RN50-BERT-Tptr	5.3	35.0	89.0	97.5	6.2	12.2	21.5	30.7	32.9	48.4	87.9	94.0	29.3	48.5	48.3	56.1
Two-Stream CLIP-Tptr (ours)	**28.3**	**64.7**	**93.3**	**98.8**	**13.7**	**24.3**	**31.2**	**41.3**	14.8	59.5	86.8	**96.2**	27.2	50.0	65.5	71.9

TABLE 3.3: **Ablations and Baselines.** Evaluation scores (mean %) for stack-block-pyramid-seq and packing-google-objects-seq tasks from 100 evaluation runs. Stacking block pyramids involves both semantic and precise spatial reasoning, whereas packing objects mostly involves semantic grounding without requiring any precise placements.

One-Stream Transporter-only is the original Transporter [311] with RGB-D input, or equivalently, the **spatial** stream of CLIPort. For all experiments, we implemented our own version of Transporter in PyTorch and did not use the modeling code provided with the original paper. Our Transporter models are also trained for 200K iterations instead of 40k iterations.

One-Stream CLIP-only is the **semantic** stream of CLIPort with RGB and language input.

One-Stream Language Transporter is Transporter [311], but the bottleneck features are conditioned with CLIP language features in a similar fashion to the **semantic** stream in CLIPort. This model performs very poorly because the high-level language features corrupt the low-level spatial features necessary for precise pick-and-place actions.

One-Stream Image-Goal Transporter is a goal-conditioned version of Transporter [243] which receives a goal-image as input. For sequential tasks with a specific order (indicated with seq in their name), we provide the goal-image from the next timestep, and for non-sequential tasks we provide the goal-image from the final timestep. The implementation follows the goal-conditioned Transporter proposed in [243], except we found that element-wise addition worked better than element-wise product for combining goal-image features with $\mathcal{Q}_{\text{place}}$ features.

Two-Stream CLIP-Transporter w/o skips is a variant of the CLIPort model without skip connections from the CLIP-ResNet encoder to the decoder layers. The results in Table 3.3 show that these skip connections are particularly important for good performance. We hypothesize that utilizing different levels of semantic information from the visual encoder – patterns,

shapes, parts, objects, and high-level concepts, is crucial for conditioning the **semantic** stream decoders.

Two-Stream RN50-BERT-Transporter is the same two-stream architecture as CLIPort, except instead of the CLIP ResNet50, we use a standard ResNet50 [161] pre-trained on ImageNet classification. And instead of the CLIP sentence encoder, we use a pretrained DistilBERT model [237] to extract language embeddings. CLIP offers the benefit of multi-modal alignment between vision and language features while not being restricted to instance segmentation or bounding box detection pipelines.

Two-Stream Untrained-Sem-Transporter uses an untrained ResNet50 and Transformer language encoder for the **semantic** stream. Even without any pre-training, the random features from the **semantic** stream somewhat help in conditioning policies. However, the performances are substantially worse than models with pre-trained multimodal features.

3.4.4 Language Ablations

In addition to the no-language baselines from Section 3.4.2, we present additional language ablations that further investigate the impact of language goals on agent performance in Figure 3.5. Following the model ablations, we evaluate single-task agents trained only on stack-block-pyramid.

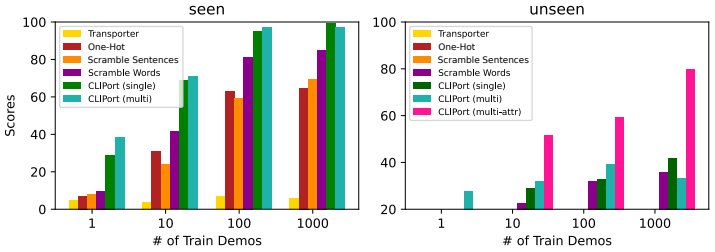


FIGURE 3.5: **Language Ablations**: Agents trained with one-hot goals, scrambled goal sentences, and goal sentences with scrambled word order.

One-Hot receives random strings as goals, but this is done consistently for each goal in the training dataset. The one-hot encoding allows agents to memorize a specific goal seen during training. However, since there is no reuse of concepts or commonality in the task specification, one-hot encodings result in zero-performance with unseen objects and goal compositions.

Scramble Sentences receives scrambled goals that are sampled from across the training dataset. For instance, *"put the red block on the blue block"* is consistently swapped with *"put the green block on the yellow block"*. Scrambling goal sentences also results in zero-performance on unseen tasks since the alignment between observations and task specification is missing in the training data.

Scramble Words receives goal sentences with words that are scrambled in a random order. For instance, *"put the red block on the blue block"* becomes *"block the put on red blue the"*. This experiment aims to determine if the agent treats the goal as a bag of words. Surprisingly, scrambled sentences result in non-zero performances for both seen and unseen tasks. However, these performances are substantially lower than CLIPort's, indicating that the order of words does matter.

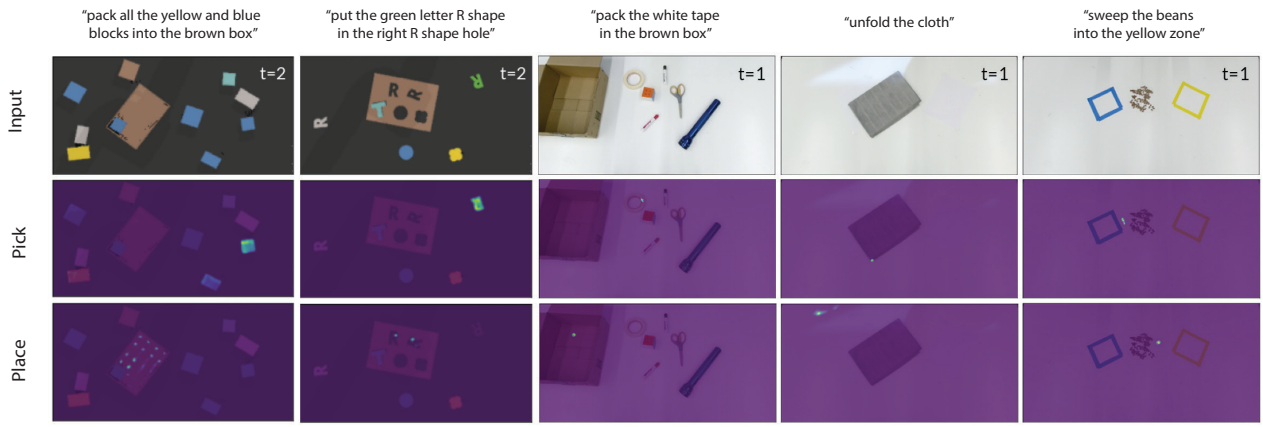


FIGURE 3.6: Affordance predictions from CLIPort (multi) models in sim (left two) and real settings (right three). More examples in Appendix B.6.

3.4.5 Real-Robot Experiments

We validated our results in hardware with a Franka Panda manipulator. See Appendix B.3 for setup details. Table 4.6 reports success rates for a multi-task model trained and evaluated on 9 real-world tasks. Due to COVID restrictions, we could not conduct large-scale user-studies, so we report on small train (5-10 demos) and test sets (5-10 runs) per task. Overall, CLIPort (multi) is effective at few-shot learning with just 179 samples, and the performances roughly correspond to those in simulated experiments, with simple block manipulation tasks achieving $\sim$ 70%. We estimate that for more robust real-world performance at least 50 to 100 training demonstrations are necessary, as evident in Figure 3.3. Interestingly, we observed that the model sometimes exploits biases in the training data instead of learning to ground instructions. For instance, in Put Blocks in Bowl, the training set consisted of only one datapoint on "yellow blocks" being placed inside a "blue bowl". This made it difficult to condition the model to place "yellow blocks" in non-blue bowls. But instances with just one or two examples where a colored block went to different colored bowls was sufficient to make the model pay attention to the language. In summary, unbiased datasets containing both a good coverage of expected skills and invariances, and a decent number of training demonstrations, are crucial for good real-world performance.

Task	# Train (Samples)	# Test	Succ. %
Stack Blocks	5 (13)	10	70.0
Put in Bowl	5 (10)	10	65.0
Pack Objects	10 (31)	10	60.0
Move Rook	4 (29)	10	70.0
Fold Cloth	9 (9)	10	57.0
Read Text	2 (26)	10	55.0
Loop Rope	4 (12)	10	60.0
Sweep Beans	5 (23)	5	60.6
Pick Cherries	4 (26)	5	75.0

TABLE 3.4: Success rates (%) of a multi-task model trained an evaluated 9 real-world tasks (see Figure 3.1). Samples indicate total image-action pairs, e.g 1 in Figure B.4.

3.5 Limitations and Risks

While CLIPort is highly capable, it is not without issues. In the following sections we discuss various limitations and risks of using CLIPort for real-world manipulation.

Balanced Datasets. CLIPort can learn generalizable policies from very few demonstrations, but it relies heavily on a balanced training dataset with a good converge of expected skills and invariances. As discussed in Section 3.4.5, the model will exploit any bias, e.g. always place "yellow blocks" inside 'blue bowls" if that is the only example of yellow blocks that it's provided with. Sometimes these biases can be hard to spot since everything (from perception to action) is trained end-to-end through demonstrations. During our real-world experiments we ended up iteratively refining some datasets after finding such biases during execution.

Hand-Eye Calibration and Closed-Loop Control. The execution of policies is sensitive to the accuracy of the hand-eye calibration. The action-space of CLIPort is 2D pixels with yaw-rotations. Translating these pixel coordinates to end-effector poses relies on carefully calibrated extrinsics between the robot's base frame and the RGB-D camera. Further, while the framework takes closed-loop actions across discrete pick-and-place timesteps, the execution of each pick and place primitive itself is open-loop. This restricts usage to mostly quasi-static tasks and leads to issues if objects move while the robot is executing a pick or place primitive. Future works could incorporate a separate visuo-servoing mechanism for more robust grasping.

Dexterous Manipulation. Extending CLIPort's action-space to 6-DOF or N-DOF control for dexterous non-quasi-static manipulation is non-trivial. The $\mathbf{SE}(2)$ action-space is one of the key factors that make Transporter and CLIPort highly data efficient. Since the actual end-effector control is abstracted away, the model can easily reason about high-level affordances at discrete timesteps, but at the price of loosing dexterity. Similarly, extending $\mathbf{SE}(2)$ equivariance to $\mathbf{SE}(3)$ equivariance is also non-trivial. Cross-correlating in voxelized 3D spaces might be expensive and slow.

Grasping Novel Objects. CLIPort has some limited capacity in grasping unseen instances of objects in one-shot or few-shot settings. While CLIP is a pure vision-language model with no understanding of affordances, actions, or physical properties, in CLIPort we fine-tune CLIP's visual representations in the **semantic** decoder layers to produce visual affordance predictions – like grasping pliers by the handle. We illustrate this in Figure 3.7 with an example of one-shot learning. Despite having seen just a single training example with pliers, CLIPort is able to correctly grasp the handles of $2/3$ unseen pliers of different shapes, sizes, and colors. The model fails in Test 3 where the instance is significantly outside the training distribution. But even so, the model is able to correctly localize the pliers among the distractor objects, and with a few more training examples it might be able to correctly grasp the instance. In contrast, RN50-BERT struggles to identify pliers with just a single example since pliers are not part of the 1000 ImageNet classes [66]. Further, without the appropriate language goal to condition the policy, e.g. when provided with a nonsensical object name like "dax", the model falls back to the most familiar object seen during training.

Grounding Complex Object Relationships. In general, CLIPort struggles with complex object-relationships that require reasoning about several objects. The model performs poorly on

assembling-kits-seq tasks that involve grounding spatial relationships like "middle" with unseen shapes and language. The model's capacity to infer these relationships purely from dense global features might be limited. Also, CLIPort cannot count objects since it does not maintain a history or belief across timesteps, thus limiting instructions to 'any' or 'all' quantifiers. Future works could explore neuro-symbolic [183] or attention-based [70] methods for better generalization to novel object-relationships.

Scope of Language Grounding. CLIPort's understanding of verb-noun phrases is tightly grounded in the demonstrations and tasks seen during training. For instance, an user could have used "sort out all the Mars bars from the pile and put them in the yellow bin" while demonstrating a task. Here the model only understands 'sort' in the context of separating something from the pile and putting it in a bin, and not in the most generic sense that is applicable in any context, like sorting numbered blocks in descending order.

Task Completion. CLIPort relies on an expert to indicate task-completion. For real-world tasks, this means the model keeps taking actions until an user stops the execution. Future works can address this issue by training a success classifier [311] to predict task completion from RGB-D observations.

Risks from Pre-Trained Models. CLIP was trained with massive amounts of "in-the-wild" image-caption pairs from the internet. This makes it prone to unchecked biases and associations [90, 24] that can be harmful to certain individuals and communities. The end-to-end framework is also vulnerable to adversarial attacks [90] that try to maliciously affect the model's behavior. These issues are further exacerbated by the fact that we use CLIP's representations to take actions with a physical robot. For safe deployment in the real-world, keeping humans in the loop – both during the training phase and while instructing the robot, might help in mitigating some of these issues and potential risks.

3.6 Discussion

We introduced CLIPort, an end-to-end framework for language-conditioned fine-grained manipulation. Our experiments, specifically with multi-task models, indicate that data-driven approaches to generalization have yet to be fully-exploited in robotics. Coupled with the right action abstraction and spatio-semantic priors, end-to-end methods can quickly learn new skills without requiring top-down pipelines that need task-specific engineering.

While CLIPort is quite capable, it is limited to 2D tabletop tasks. 2D pick-and-place frameworks are quite robust, and already well established in industry, but real-world manipulation tasks are rarely confined to tabletop planes. Can we take the key principle of CLIPort, i.e., language-conditioned action detection, and extend it to 6-DoF manipulation? This is what we investigate in the next chapter.

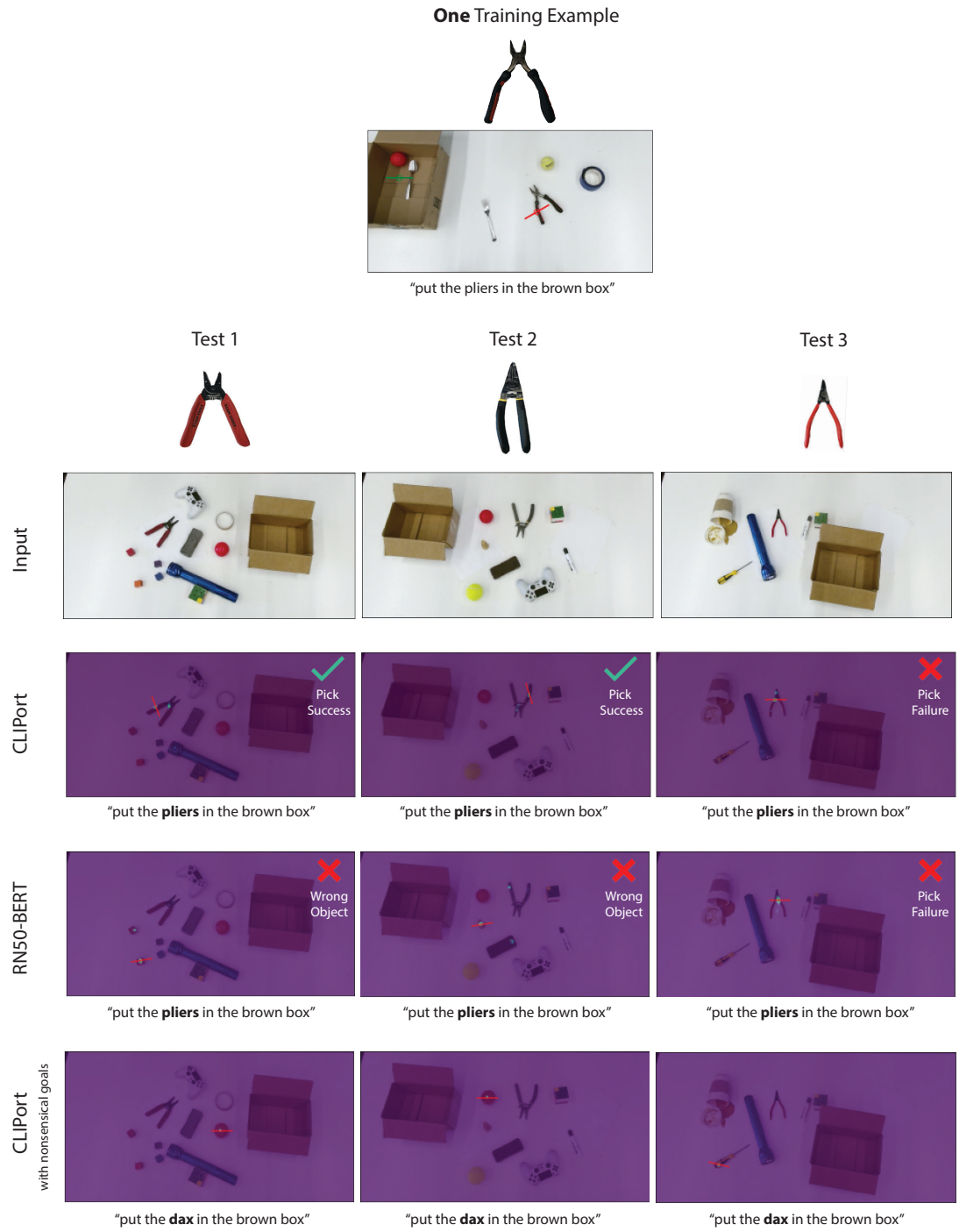


FIGURE 3.7: **One-Shot Learning**. Selected examples of grasping pliers with CLIPort, RN50-BERT, and CLIPort with nonsensical goals.

Chapter 4

Grounding Language in Precise 3D Actions

Robots and humans live in 3D environments, not 2D image grids. While CLIPort's ability to ground language in precise actions can be scaled-up to thousands of manipulation tasks, it is limited to 2D pick-and-place settings with an action-space of $x, y, \theta_{\text{yaw}}$. Real-world manipulation often goes beyond 2D to 6-DoF settings with an action-space of $x, y, z, \theta_{\text{yaw}}, \phi_{\text{pitch}}, \psi_{\text{roll}}$.

In this chapter, we address this limitation by extending CLIPort's language-conditioned action detector to 6-DoF manipulation by utilizing 3D voxel grids (or volumetric pixel grids) instead of 2D images as the input and output space. In contrast to images, voxel inputs are much higher dimensional than images, and tasks often involve attending to different parts of the large input space. Luckily, one recent architecture has been shown to be quite effective in modeling long-range dependencies: *Transformers*.

We present PERACT, a Transformer-based agent for language-conditioned behavior cloning. PERACT encodes language goals and RGB-D voxel observations with a PerceiverIO [120] based Transformer (an architecture designed for long-range input with limited resource usage), and outputs discretized actions by "detecting the next best voxel action". Unlike frameworks that operate on 2D images, the voxelized 3D observation and action space provides a strong structural prior for efficiently learning 6-DoF policies. With this formulation, we train a single multi-task Transformer for 18 RLBench [127] tasks with 249 variations and 7 real-world tasks with 18 variations from just a few demonstrations per task. Our results show that PERACT significantly outperforms unstructured image-to-action agents and 3D ConvNet baselines for a wide range of tabletop tasks.

This chapter was previously published as Shridhar et al. [251]. The code, data, pre-trained models, and implementation tutorials are available at peract.github.io.

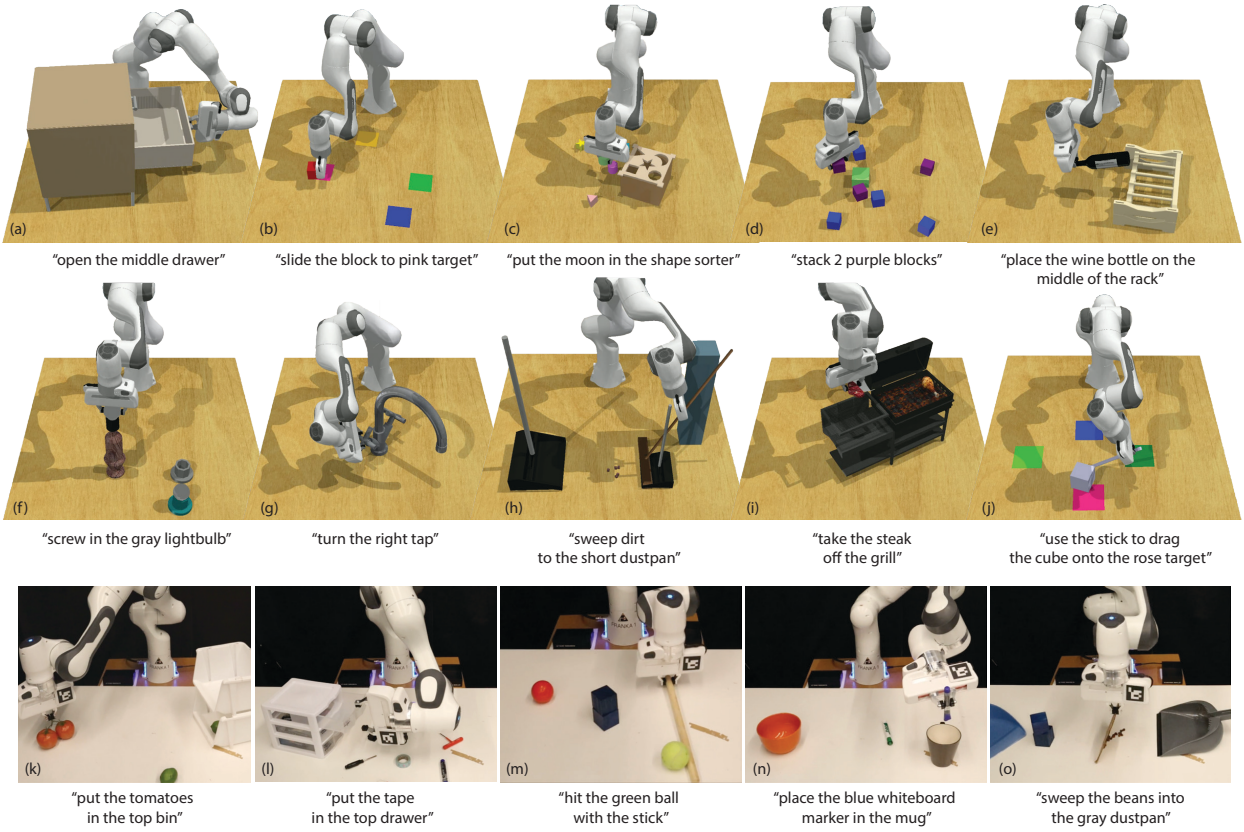


FIGURE 4.1: **Language-Conditioned Manipulation Tasks:** PERACT is a language-conditioned multi-task agent capable of imitating a wide range of 6-DoF manipulation tasks. We conduct experiments on 18 simulated tasks in RLBench [127] (a-j; only 10 shown), with several pose and semantic variations. We also demonstrate our approach with a Franka Panda on 7 real-world tasks (k-o; only 5 shown) with a multi-task agent trained with just 53 demonstrations.

4.1 Overview

Transformers [281] have become prevalent in natural language processing and computer vision. By framing problems as sequence modeling tasks, and training on large amounts of diverse data, Transformers have achieved groundbreaking results in several domains [37, 71, 137, 282]. Even in domains that do not conventionally involve sequence modeling [48, 45], Transformers have been adopted as a *general* architecture [228]. But in robotic manipulation, data is both limited and expensive. Can we still bring the power of Transformers to 6-DoF manipulation with the right problem formulation?

Language models operate on sequences of tokens [69], and vision transformers operate on sequences of image patches [71]. While pixel transformers [119, 120] exist, they are not as data efficient as approaches that use convolutions or patches to exploit the 2D structure of images. Thus, while Transformers may be domain agnostic, they still require the right problem formulation to be data efficient. A similar efficiency issue is apparent in behavior-cloning (BC) agents that directly map 2D images to 6-DoF actions. Agents like Gato [228] and BC-Z [129, 5] have shown impressive multi-task capabilities, but they require several weeks or even months

of data collection. In contrast, recent works in reinforcement-learning like C2FARM [128] construct a voxelized observation and action space to efficiently learn visual representations of 3D actions with 3D ConvNets. Similarly, in this work, we aim to exploit the 3D structure of *voxel patches* for efficient 6-DoF behavior-cloning with Transformers (analogous to how vision transformers [71] exploit the 2D structure of image patches).

To this end, we present PERACT (short for PERCEIVER-ACTOR), a language-conditioned BC agent that can learn to imitate a wide variety of 6-DoF manipulation tasks with just a few demonstrations per task. PERACT encodes a sequence of RGB-D voxel patches and predicts discretized translations, rotations, and gripper actions that are executed with a motion-planner in an observe-act loop. PERACT is essentially a classifier trained with supervised learning to *detect actions* akin to prior work like CLIPort [250, 311], except our observations and actions are represented with 3D voxels instead of 2D image pixels. Voxel grids are less prevalent than images in end-to-end BC approaches often due to scaling issues with high-dimensional inputs. But in PERACT, we use a Perceiver[1] Transformer [119] to encode very high-dimensional input of up to 1 million voxels with only a small set of latent vectors. This voxel-based formulation provides a strong structural prior with several benefits: a natural method for fusing multi-view observations, learning robust action-centric[2] representations [89, 36], and enabling data augmentation in 6-DoF – all of which help learn generalizable skills by focusing on *diverse* rather than narrow multi-task data.

To study the effectiveness of this formulation, we conduct large-scale experiments in the RLBench [127] environment. We train a single multi-task agent on 18 diverse tasks with 249 variations that involve a range of prehensile and non-prehensile behaviors like placing wine bottles on a rack and dragging objects with a stick (see Figure 4.1 a-j). Each task also includes several pose and semantic variations with objects that differ in placement, color, shape, size, and category. Our results show that PERACT significantly outperforms image-to-action agents (by 34×) and 3D ConvNet baselines (by 2.8×), without using any explicit representations of instance segmentations, object poses, memory, or symbolic states. We also validate our approach on a Franka Panda with a multi-task agent trained *from scratch* on 7 real-world tasks with a total of just 53 demonstrations (see Figure 4.1 k-o).

4.2 Related Work

Vision for Manipulation. Traditionally, methods in robot perception have used explicit "object" representations like instance segmentations, object classes, poses [106, 292, 322, 316, 67, 294]. Such methods struggle with deformable and granular items like cloths and beans that are hard to represent with geometric models or segmentations. In contrast, recent methods [314, 311, 250, 261] learn action-centric representations without any "objectness" assumptions, but they are limited to top-down 2D settings with simple pick-and-place primitives. In 3D, James et al. proposed C2FARM [128], an action-centric reinforcement learning (RL) agent with a coarse-to-fine-grain 3D-UNet backbone. The coarse-to-fine-grain scheme has a limited

[1] Throughout the thesis we refer to PerceiverIO [119] as Perceiver for brevity.
[2] Action-centric refers to a system that learns perceptual representations of actions; see Figure 4.11.

receptive field that cannot look at the entire scene at the finest level. In contrast, PERACT learns action-centric representations with a global-receptive field through a Transformer backbone. Also, PERACT does BC instead of RL, which enables us to easily train a multi-task agent for several tasks by conditioning it with language goals.

End-to-End Manipulation approaches [140, 290, 161, 79] make the least assumptions about objects and tasks, but are often formulated as an image-to-action prediction task. Training directly on RGB images for 6-DoF tasks is often inefficient, generally requiring several demonstrations or episodes just to learn basic skills like rearranging objects. In contrast, PERACT uses a voxelized observation and action space, which is dramatically more efficient and robust in 6-DoF settings. While other works in 6-DoF grasping [260, 200, 198, 296, 4, 255] have used RGB-D and pointcloud input, they have not been applied to sequential tasks or used with language-conditioning. Another line of work tackles data inefficiency by using pre-trained image representations [250, 203, 308] to bootstrap BC. Although our framework is trained from scratch, such pre-training approaches could be integrated together in future works for even greater efficiency and generalization to unseen objects.

Transformers for Agents and Robots. Transformers have become the prevalent architecture in several domains. Starting with NLP [281, 37, 167], recently in vision [71, 168], and even RL [45, 131, 160]. In robotics, Transformers have been applied to assistive teleop [54], legged locomotion [297], path-planning [41, 134], imitation learning [64, 145], morphology controllers [97], spatial rearrangement [165], and grasping [99]. Transformers have also achieved impressive results in multi-domain settings like in Gato [228] where a single Transformer was trained on 16 domains such as captioning, language-grounding, robotic control etc. However, Gato relies on extremely large datasets like 15K episodes for block stacking and 94K episodes for Meta-World [307] tasks. Our approach might complement agents like Gato, which could use our 3D formulation for greater efficiency and robustness.

Language Grounding for Manipulation. Several works have proposed methods for grounding language in robot actions [249, 186, 32, 195, 28, 277, 101, 50, 29, 219, 271, 207]. However, these methods use disentangled pipelines for perception and action, with the language primarily being used to guide perception [26]. Recently, a number of end-to-end approaches [5, 129, 202, 188, 174] have been proposed for conditioning BC agents with language instructions. These methods require thousands of human demos or autonomous episodes that are collected over several days or even months. In contrast, PERACT can learn robust multi-task policies with just a few minutes of training data. For benchmarking, several simulation environments exist [189, 311, 307], but we use RLBench [127] for its diversity of 6-DoF tasks and ease of generating demonstrations with templated language goals.

Voxel Representations. Voxel-based representations have been used in several domains that specifically benefit from 3D understanding. Like in object detection [182, 104], object search [319], and vision-language grounding [31, 56], voxel maps have been used to build persistent scene

representations [258]. In Neural Radiance Fields (NeRFs), voxel feature grids have dramatically reduced training and rendering times [199, 239]. Similarly, other works in robotics have used voxelized representations to embed viewpoint-invariance for driving [156] and manipulation [279]. The use of latent vectors in Perceiver [119] is broadly related to voxel hashing [208] from computer graphics. Instead of using a location-based hashing function to map voxels to fixed size memory, PerceiverIO uses cross-attention to map the input to fixed size latent vectors, which are trained end-to-end. Another major difference is the treatment of unoccupied space. In graphics, unoccupied space does not affect rendering, but in PERACT, unoccupied space is where a lot of "action detections" happen. Thus the relationship between unoccupied and occupied space, i.e., scene, objects, robot, is crucial for learning action representations.

Long-Context and Latent-Space Transformers. Several approaches have been proposed for extending Transformers to longer context lengths [268]. Latent-space Transformers that use fixed-size latents instead of the full context, are one such approach [120, 93]. There is no clear winner in terms of trade-offs between speed, memory, and performance. However, latent-space methods have achieved compelling results in object detection [39] and slot-attention based object discovery [170].

4.3 PERCEIVER-ACTOR

PERACT is a language-conditioned behavior-cloning agent for 6-DoF manipulation. The key idea is to learn perceptual representations of actions conditioned on language goals. Given a voxelized reconstruction of a scene, we use a Perceiver Transformer [119] to learn per-voxel features. Despite the extremely large input space (100^3), Perceiver uses a small set of latent vectors to encode the input. The per-voxel features are then used to predict the next best action in terms of discretized translation, rotation, and gripper state at each timestep. PERACT relies purely on the current observation to determine what to do next in sequential tasks. See Figure 4.2 for an overview.

Section 4.3.1 and Section 4.3.2 describe our dataset setup. Section 4.3.3 describes our problem formulation with PERACT, and Section 4.3.4 provides details on training PERACT.

4.3.1 Demonstrations

We assume access to a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, \ldots, \zeta_n\}$ of n expert demonstrations, each paired with English language goals $\mathcal{G} = \{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_n\}$. These demonstrations are collected by an expert with the aid of a motion-planner to reach intermediate poses. Each demonstration ζ is a sequence of continuous actions $\mathcal{A} = \{a_1, a_2, \ldots, a_t\}$ paired with observations $\mathcal{O} = \{\tilde{o}_1, \tilde{o}_2, \ldots \tilde{o}_t\}$. An action a consists of the 6-DoF pose,

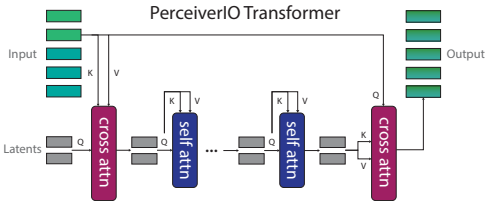


FIGURE 4.3: **Perceiver Transformer Architecture.** Perceiver is a latent-space transformer. Q, K, V represent queries, keys, and values, respectively.

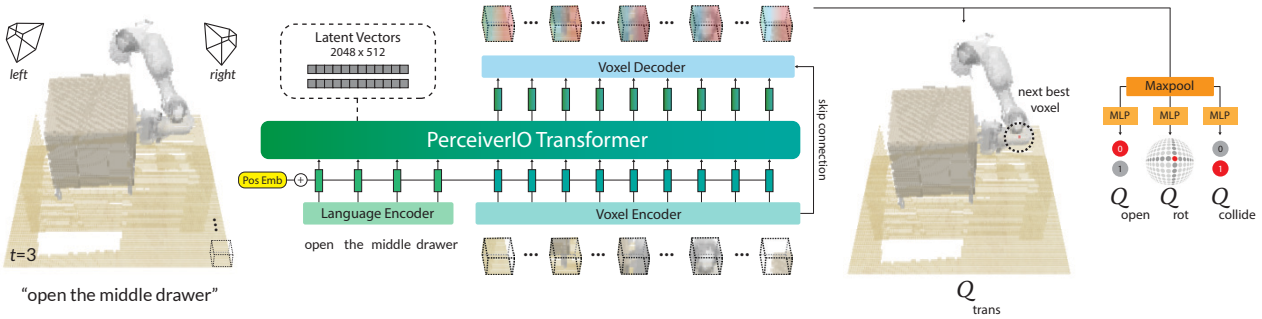


FIGURE 4.2: **PERACT Overview.** PERACT is a language-conditioned behavior-cloning agent trained with supervised learning to *detect actions*. PERACT takes as input a language goal and a voxel grid reconstructed from RGB-D sensors. The voxels are split into 3D patches, and the language goal is encoded with a pre-trained language model. These language and voxel features are appended together as a sequence and encoded with a Perceiver transformer [119]. Despite the extremely long input sequence, Perceiver uses a small set of latent vectors to encode the input (see Figure 4.3 for an illustration). These encodings are upsampled back to the original voxel dimensions with a decoder and reshaped with linear layers to predict a discretized translation, rotation, gripper open, and collision avoidance action. This action is executed with a motion-planner after which the new observation is used to predict the next discrete action in an observe-act loop until termination.

gripper open state, and whether the motion-planner used collision avoidance to reach an intermediate pose: $a = \{a_{\text{pose}}, a_{\text{open}}, a_{\text{collide}}\}$. An observation $\tilde{o}$ consists of RGB-D images from any number of cameras. We use four cameras for simulated experiments $\tilde{o}_{\text{sim}} = \{o_{\text{front}}, o_{\text{left}}, o_{\text{right}}, o_{\text{wrist}}\}$, but just a single camera for real-world experiments $\tilde{o}_{\text{real}} = \{o_{\text{front}}\}$.

Following prior work by James et al. [128], we construct a structured observation and action space through keyframe extraction and voxelization.

4.3.2 Keyframes and Voxelization

Training our agent to directly predict continuous actions is inefficient and noisy. So instead, for each demonstration ζ, we extract a set of keyframe actions $\{\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_m\} \subset \mathcal{A}$ that capture bottleneck end-effector poses [133] in the action sequence with a simple heuristic: an action is a keyframe if (1) the joint-velocities are near zero and (2) the gripper open state has not changed. Each datapoint in the demonstration ζ can then be cast as a "predict the next (best) keyframe action" task [128, 125, 164]. This process is illustrated in Figure C.2, where $\mathbf{k}_1$ and $\mathbf{k}_2$ are two keyframes that were extracted from the heuristic. The orange circles indicate datapoints whose RGB-D observations are paired with the next keyframe action.

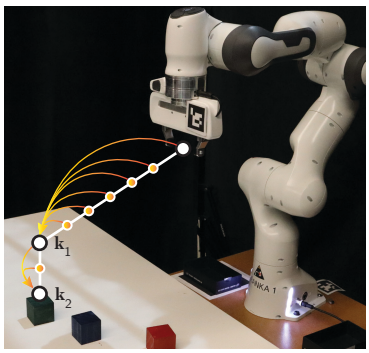


FIGURE 4.4: **Keyframes and Demo Augmentation.**

To learn action-centric representations [89] in 3D, we use a voxel grid [196, 235] to represent both the observation and action space. The observation voxels $\mathbf{v}$ are reconstructed from RGB-D observations $\tilde{o}$ fused through triangulation $\tilde{o} \Rightarrow \mathbf{v}$ from known camera extrinsics and intrinsics.

By default, we use a voxel grid of 100^3, which corresponds to a volume of 1.0m^3 in metric scale. The keyframe actions $\mathbf{k}$ are discretized such that training our BC agent can be formulated as a "next best action" classification task [128]. Translation is simply the closest voxel to the center of the gripper fingers. Rotation is discretized into 5 degree bins for each of the three rotation axes. Gripper open state is a binary value. Collide is also a binary value that indicates if the motion-planner should avoid everything in the voxel grid or nothing at all; switching between these two modes of collision avoidance is crucial as tasks often involve both contact based (e.g., pulling the drawer open) and non-contact based motions (e.g., reaching the handle without colliding into anything).

4.3.3 PERACT Agent

PERACT is a Transformer-based [281] agent that takes in a voxel observation and language goal $(\mathbf{v}, \mathbf{l})$, and outputs a discretized translation, rotation, and gripper open action. This action is executed with a motion-planner, after which this process is repeated until the goal is reached.

The language goal $\mathbf{l}$ is encoded with a pre-trained language model. We use CLIP's [224] language encoder, but any pre-trained language model would suffice [129, 174]. Our choice of CLIP opens up possibilities for future work to use pre-trained vision features that are aligned with the language for better generalization to unseen semantic categories and instances [250].

The voxel observation $\mathbf{v}$ is split into 3D patches of size 5^3 (akin to vision-transformers like ViT [71]). In implementation, these patches are extracted with a 3D convolution layer with a kernel-size and stride of 5, and then flattened into a sequence of voxel encodings. The language encodings are fine-tuned with a linear layer and then appended with the voxel encodings to form the input sequence. We also add learned positional embeddings to the sequence to incorporate voxel and token positions.

The input sequence of language and voxel encodings is extremely long. A standard Transformer with $\mathcal{O}(n^2)$ self-attention connections and an input of $(100/5)^3 = 8000$ patches is hard to fit on the memory of a commodity GPU. Instead, we use the Perceiver [119] Transformer. Perceiver is a latent-space Transformer, where instead of attending to the entire input, it first computes cross-attention between the input and a much smaller set of latent vectors (which are randomly initialized and trained). These latents are encoded with self-attention layers, and for the final output, the latents are again cross-attended with the input to match the input-size. See Figure 4.3 for an illustration. By default, we use 2048 latents of dimension 512 : $\mathbb{R}^{2048\times512}$, but in Section 4.4.4 we experiment with different latent sizes.

The Perceiver Transformer uses 6 self-attention layers to encode the latents and outputs a sequence of patch encodings from the output cross-attention layer. These patch encodings are upsampled with a 3D convolution layer and tri-linear upsampling to decode 64-dimensional voxel features. The decoder includes a skip-connection from the encoder (like in UNets [233]). The per-voxel features are then used to predict discretized actions [128]. For translation, the voxel features are reshaped into the original voxel grid (100^3) to form a 3D $\mathcal{Q}$-function of action-values. For rotation, gripper open, and collide, the features are max-pooled and then decoded with linear layers to form their respective $\mathcal{Q}$-function. The best action $\mathcal{T}$ is chosen by simply maximizing the $\mathcal{Q}$-functions:

$$\begin{aligned} \mathcal{T}_{\text{trans}} &= \underset{(x,y,z)}{\text{argmax}}\ \mathcal{Q}_{\text{trans}}((x,y,z) \mid \mathbf{v}, \mathbf{l}), & \mathcal{T}_{\text{rot}} &= \underset{(\psi,\theta,\phi)}{\text{argmax}}\ \mathcal{Q}_{\text{rot}}((\psi,\theta,\phi) \mid \mathbf{v}, \mathbf{l}), \\ \mathcal{T}_{\text{open}} &= \underset{\omega}{\text{argmax}}\ \mathcal{Q}_{\text{open}}(\omega \mid \mathbf{v}, \mathbf{l}), & \mathcal{T}_{\text{collide}} &= \underset{\kappa}{\text{argmax}}\ \mathcal{Q}_{\text{collide}}(\kappa \mid \mathbf{v}, \mathbf{l}), \end{aligned} \tag{4.1}$$

where (x, y, z) is the voxel location in the grid, (ψ, θ, ϕ) are discrete rotations in Euler angles, ω is the gripper open state and κ is the collide variable. See Figure 4.8 for examples of $\mathcal{Q}$-predictions.

4.3.4 Training Details

PERACT is trained through supervised learning with discrete-time input-action tuples from a dataset of demonstrations. These tuples are composed of voxel observations, language goals, and keyframe actions $\{(\mathbf{v}_1, \mathbf{l}_1, \mathbf{k}_1), (\mathbf{v}_2, \mathbf{l}_2, \mathbf{k}_2), \ldots\}$. During training, we randomly sample a tuple and supervise the agent to predict the keyframe action $\mathbf{k}$ given the observation and goal $(\mathbf{v}, \mathbf{l})$. For translation, the ground-truth action is represented as a one-hot voxel encoding $Y_{\text{trans}} : \mathbb{R}^{H \times W \times D}$. Rotations are also represented with a one-hot encoding per rotation axis with R rotation bins $Y_{\text{rot}} : \mathbb{R}^{(360/R) \times 3}$ ($R = 5$ degrees for all experiments). Similarly, open and collide variables are binary one-hot vectors $Y_{\text{open}} : \mathbb{R}^2$, $Y_{\text{collide}} : \mathbb{R}^2$. The agent is trained with cross-entropy loss:

$$\mathcal{L}_{\text{total}} = -\mathbb{E}_{Y_{\text{trans}}}[\log \mathcal{V}_{\text{trans}}] - \mathbb{E}_{Y_{\text{rot}}}[\log \mathcal{V}_{\text{rot}}] - \mathbb{E}_{Y_{\text{open}}}[\log \mathcal{V}_{\text{open}}] - \mathbb{E}_{Y_{\text{collide}}}[\log \mathcal{V}_{\text{collide}}],$$

where $\mathcal{V}_{\text{trans}} = \text{softmax}(\mathcal{Q}_{\text{trans}}((x, y, z)|\mathbf{v}, \mathbf{l}))$, $\mathcal{V}_{\text{rot}} = \text{softmax}(\mathcal{Q}_{\text{rot}}((\psi, \theta, \phi)|\mathbf{v}, \mathbf{l}))$, $\mathcal{V}_{\text{open}} = \text{softmax}(\mathcal{Q}_{\text{open}}(\omega|\mathbf{v}, \mathbf{l}))$, $\mathcal{V}_{\text{collide}} = \text{softmax}(\mathcal{Q}_{\text{collide}}(\kappa|\mathbf{v}, \mathbf{l}))$ respectively. For robustness, we also augment $\mathbf{v}$ and $\mathbf{k}$ with translation and rotation perturbations.

By default, we use a voxel grid size of 100^3. We conducted validation tests by replaying expert demonstrations with discretized actions to ensure that 100^3 is a sufficient resolution for execution. The agent was trained with a batch-size of 16 on 8 NVIDIA V100 GPUs for 16 days (600K iterations). We use the LAMB [300] optimizer following Perceiver [119].

For multi-task training, we simply sample input-action tuples from all tasks in the dataset. To ensure that tasks with longer horizons are not over-represented during sampling, each batch contains a uniform distribution of tasks. That is, we first uniformly sample a set of tasks of batch-size length, then pick a random input-action tuple for each of the sampled tasks. With this strategy, longer-horizon tasks need more training steps for full coverage of input-action pairs, but all tasks are given equal weighting during gradient updates.

4.4 Experiments

We perform experiments to answer the following questions: (1) How effective is PERACT compared to unstructured image-to-action frameworks and standard architectures like 3D ConvNets? And what are the factors that affect PERACT's performance? (2) Is the global receptive

field of Transformers actually beneficial over methods with local receptive fields? (3) Can PERACT be trained on real-world tasks with noisy data? (4) Can PERACT be quickly fine-tuned on new tasks? And does the pre-training help?

4.4.1 Simulation Setup

We conduct our primary experiments in simulation for the sake of reproducibility and benchmarking.

Environment. The simulation is set in CoppelaSim [230] and interfaced through PyRep [126]. All experiments use a Franka Panda robot with a parallel gripper. The input observations are captured from four RGB-D cameras positioned at the front, left shoulder, right shoulder, and on the wrist, as shown in Figure 4.5. All cameras are noiseless and have a resolution of 128×128.

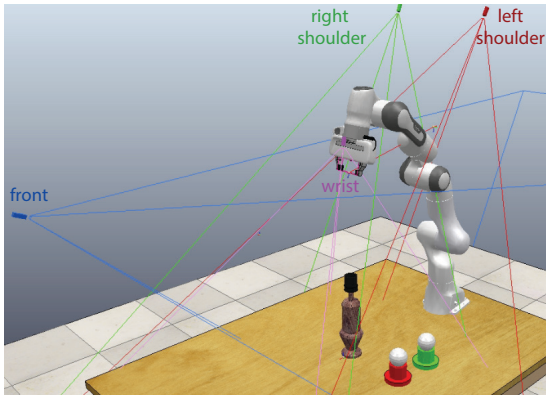


FIGURE 4.5: **Simulated Setup.** The four camera setup: front, left shoulder, right shoulder, and on the wrist.

Language-Conditioned Tasks. We train and evaluate on 18 RLBench [127] tasks. See peract.github.io for examples, and Table 4.1 for details on individual tasks. Each task includes several variations, ranging from 2-60 possibilities, e.g., in the `stack blocks` task, *"stack 2 red blocks"* and *"stack 4 purple blocks"* are two variants. These variants are randomly sampled during data generation, but kept consistent during evaluations for one-to-one comparisons. Some RLBench tasks were modified to include additional variations to stress-test multi-task and language-grounding capabilities. There are a total of 249 variations across 18 tasks, and the number of extracted keyframes range from 2-17. All keyframes from an episode have the same language goal, which is constructed from templates (but human-annotated for real-world tasks). Note that in all experiments, we do not test for generalization to unseen objects, i.e., our train and test objects are the same. However during test time, the agent has to handle novel object poses, randomly sampled goals, and randomly sampled scenes with different semantic instantiations of object colors, shapes, sizes, and categories. The focus here is to evaluate the performance of a single multi-task agent trained on all tasks and variants.

Setup. Our simulated experiments are set in RLBench [127]. We select 18 out of 100 tasks that involve at least two or more variations to evaluate the multi-task capabilities of agents. While PERACT could be easily applied to more RLBench tasks, in our experiments, we were specifically interested grounding diverse language instructions, rather than learning one-off policies for single-variation tasks like *"[always] take off the saucepan lid"*. Some tasks were modified to

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

TABLE 4.1: **Language-Conditioned Tasks in RLBench** [127]. Variation type, number of variations, average number of keyframes in a demonstration, and templates used to generate language goals for all tasks used in our experiments.

include additional variations. See Table 4.1 for an overview. We report average keyframes extracted from the method described in Section 4.3.2.

Variations. Task variations include randomly sampled colors, sizes, shapes, counts, placements, and categories of objects. The set of colors include 20 instances: `colors` = {`red`, `maroon`, `lime`, `green`, `blue`, `navy`, `yellow`, `cyan`, `magenta`, `silver`, `gray`, `orange`, `olive`, `purple`, `teal`, `azure`, `violet`, `rose`, `black`, `white`}. The set of sizes include 2 instances: `sizes` = {`short`, `tall`}. The set of shapes include 5 instances: `shapes` = {`cube`, `cylinder`, `triangle`, `star`, `moon`}. The set of counts include 3 instances: `counts` = {1, 2, 3}. The placements and object categories are specific to each task. For instance, `open drawer` has 3 placement locations: `top`, `middle`, and `bottom`, and `put in cupboard` includes 9 YCB objects. In addition to these semantic variations, objects are placed on the tabletop at random poses. Some large objects like drawers have constrained pose variations [127] to ensure that manipulating them is kinematically feasible with the Franka arm.

Evaluation Metric. Each multi-task agent is evaluated independently on all 18 tasks. Evaluations are scored either 0 for failures or 100 for complete successes. There are no partial credits. We report average success rates on 25 evaluation episodes per task ($25 \times 18 = 450$ total episodes) for agents trained with $n = 10, 100$ demonstrations per task. During evaluation, an agent keeps taking actions until an oracle indicates task-completion or reaches a maximum of 25 steps.

	open drawer		slide block		sweep to dustpan		meat off grill		turn tap		put in drawer		close jar		drag stick		stack blocks	
Method	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
Image-BC (CNN)	4	4	4	0	0	0	0	0	20	8	0	8	0	0	0	0	0	0
Image-BC (ViT)	16	0	8	0	8	0	0	0	24	16	0	0	0	0	0	0	0	0
C2FARM-BC	28	20	12	16	4	0	40	20	60	68	12	4	28	24	**72**	24	4	0
PERACT (no lan)	20	28	8	12	20	16	40	48	36	60	16	16	16	12	48	60	0	0
PERACT	**68**	**80**	**32**	**72**	**72**	**56**	**68**	**84**	**72**	**80**	**16**	**68**	**32**	**60**	36	**68**	**12**	**36**

	screw bulb		put in safe		place wine		put in cupboard		sort shape		push buttons		insert peg		stack cups		place cups	
	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
Image-BC (CNN)	0	0	0	4	0	0	0	0	0	0	4	0	0	0	0	0	0	0
Image-BC (ViT)	0	0	0	0	4	0	4	0	0	0	16	0	0	0	0	0	0	0
C2FARM-BC	12	8	0	12	**36**	8	**4**	0	8	8	**88**	**72**	0	**4**	0	0	0	0
PERACT (no lan)	0	**24**	8	20	8	**20**	0	0	0	0	60	68	4	0	0	0	0	0
PERACT	**28**	**24**	**16**	**44**	20	12	0	**16**	**16**	**20**	56	48	**4**	0	0	0	0	0

TABLE 4.2: **Multi-Task Test Results.** Success rates (mean %) of various multi-task agents tasks trained with either 10 or 100 demonstrations per task and evaluated on 25 episodes per task. Each evaluation episode is scored either a 0 for failure or 100 for succces. PERACT outperforms C2FARM-BC [128], the most competitive baseline, with an average improvement of 1.33× with 10 demos and 2.83× with 100 demos.

4.4.2 Simulation Results

Table 4.2 reports success rates of multi-task agents trained on all 18 tasks. We could not investigate single-task agents due to resource constraints of training 18 individual agents.

Baseline Methods. We study the effectiveness of our problem formulation by benchmarking against two language-conditioned baselines: Image-BC and C2FARM-BC. Image-BC is an image-to-action agent similar to BC-Z [129]. Following BC-Z, we use FiLM [220] for conditioning with CLIP [224] language features, but the vision encoders take in RGB-D images instead of just RGB. We also study both CNN and ViT vision encoders. C2FARM-BC is a 3D fully-convolutional network by James et al. [128] that has achieved state-of-the-art results on RLBench tasks. Similar to our agent, C2FARM-BC also detects actions in a voxelized space, however it uses a coarse-to-fine-grain scheme to detect actions at two-levels of voxelization: 32^3 voxels with a 1^3m grid, and 32^3 voxels with a 0.15^3m grid after "zooming in" from the first level. Note that at the finest level, C2FARM-BC has a higher resolution (0.47cm) than PERACT (1cm). We use the same 3D ConvNet architecture as James et al. [128], but instead of training it with RL, we do BC with cross-entropy loss (from Section 4.3.4). We also condition it with CLIP [224] language features at the bottleneck like in LingUNets [191, 250].

Multi-Task Performance. Table 4.2 compares the performance of Image-BC and C2FARM-BC against PERACT. With insufficient demonstrations, Image-BC has near zero performance on most tasks. Image-BC is disadvantaged with single-view observations and has to learn hand-eye coordination from scratch. In contrast, PERACT's voxel-based formulation naturally allows for integrating multi-view observations, learning 6-DoF action representations,

and data-augmentation in 3D, all of which are non-trivial to achieve in image-based methods. C2FARM-BC is the most competitive baseline, but it has a limited receptive field mostly because of the coarse-to-fine-grain scheme and partly due to the convolution-only architecture. PERACT outperforms C2FARM-BC in 25/36 evaluations in Table 4.2 with an average improvement of 1.33× with 10 demonstrations and 2.83×with 100 demonstrations. For a number of tasks, C2FARM-BC actually performs worse with more demonstrations, likely due to insufficient capacity. Since additional training demonstrations include additional task variants to optimize for, they might end up hurting performance.

In general, 10 demonstrations are sufficient for PERACT to achieve $>$ 65% success on tasks with limited variations like `open drawer` (3 variations). But tasks with more variations like `stack blocks` (60 variations) need substantially more data, sometimes to simply cover all possible concepts like *"teal color block"* that might have not appeared in the training data. See the simulation rollouts in the supplementary video to get a sense of the complexity of these evaluations. For three tasks: `insert peg`, `stack cups`, and `place cups`, all agents achieve near zero success. These are very high-precision tasks where being off by a few centimeters or degrees could lead to unrecoverable failures. But in Section 4.4.5 we find that training single-task agents, specifically for these tasks, slightly alleviates this issue.

Ablations. Table 4.2 reports PERACT w/o Lang, an agent without any language conditioning. Without a language goal, the agent does not know the underlying task and performs at chance. We also report additional ablation results on the `open drawer` task in Figure 4.6. To summarize these results: (1) the skip connection helps train the agent slightly faster, (2) the Perceiver Transformer is crucial for achieving good performance with the global receptive field, and (3) extracting good keyframes actions is essential for supervised training as randomly chosen or fixed-interval keyframes lead to zero-performance.

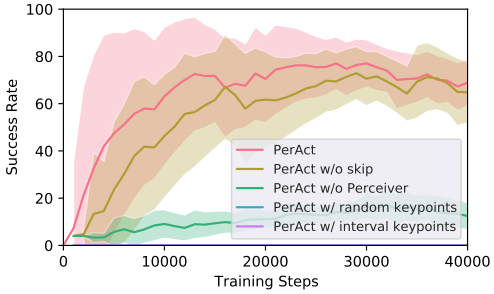


FIGURE 4.6: **Ablation Experiments.** Success rate of PERACT after ablating key components.

4.4.3 Global vs. Local Receptive Fields

To further investigate our Transformer agent's global receptive field, we conduct additional experiments on the `open drawer` task. The `open drawer` task has three variants: *"open the top drawer"*, *"open the middle drawer"*, and *"open the bottom drawer"*, and with a limited receptive field it is hard to distinguish the drawer handles, which are all visually identical. Figure 4.7 reports PERACT and C2FARM-BC agents trained with 100 demonstrations. Although the `open drawer` tasks can be solved with fewer demonstrations, here we

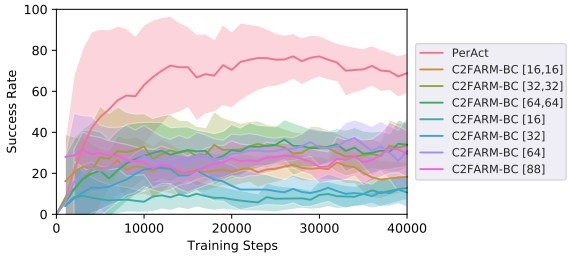


FIGURE 4.7: **Global vs. Local Receptive Field Experiments.** Success rates of PERACT against various C2FARM-BC [128] baselines

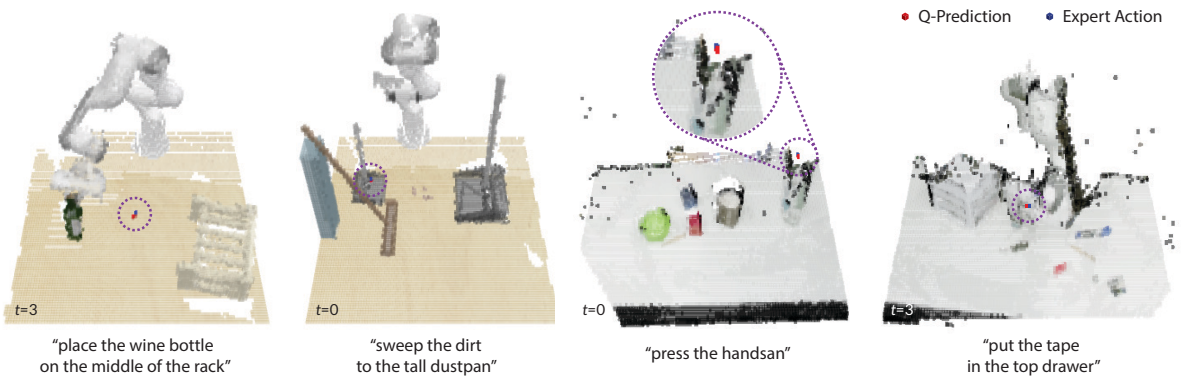


FIGURE 4.8: **Q-Prediction Examples.** Qualitative examples of translation Q-Predictions from PERACT along with expert actions, highlighted with dotted-circles. The left two are simulated tasks, and the right two are real-world tasks. See Figure 4.11 for more examples.

want to ensure that insufficient data is not an issue.

We include several versions of C2FARM-BC with different voxelization schemes. For instance, $[16, 16]$ indicates two levels of 16^3 voxel grids at 1m^3 and 0.15m^3, respectively. And $[64]$ indicates a single level of a 64^3 voxel grid without the coarse-to-fine-grain scheme. PERACT is the only agent that achieves $> 70\%$ success, whereas all C2FARM-BC versions perform at chance

	open drawer	slide block	sweep to dustpan	meat off grill	turn tap	put in drawer	close jar	drag stick	stack blocks
PERACT	80	72	56	84	80	68	60	68	36
PERACT w/o Rot Aug	92	72	56	92	96	60	56	100	8
PERACT 4096 latents	84	88	44	68	84	48	48	84	12
PERACT 1024 latents	84	48	52	84	84	52	32	92	12
PERACT 512 latents	92	84	48	100	92	32	32	100	20
PERACT 64^3 voxels	88	72	80	60	84	36	40	84	32
PERACT 32^3 voxels	28	44	100	60	72	24	0	24	0
PERACT 7^3 patches	72	48	96	92	76	76	36	96	32
PERACT 9^3 patches	68	64	56	52	96	56	36	92	20

	screw bulb	put in safe	place wine	put in cupboard	sort shape	push buttons	insert peg	stack cups	place cups
PERACT	24	44	12	16	20	48	0	0	0
PERACT w/o Rot Aug	20	32	48	8	8	56	8	4	0
PERACT 4096 latents	32	44	52	8	12	72	4	4	0
PERACT 1024 latents	24	32	36	8	20	40	8	4	0
PERACT 512 latents	48	40	36	24	16	32	12	0	4
PERACT 64^3 voxels	24	48	44	12	4	32	0	4	0
PERACT 32^3 voxels	12	20	52	0	0	60	0	0	0
PERACT 7^3 patches	8	48	76	0	12	16	0	0	0
PERACT 9^3 patches	12	36	72	12	0	20	0	0	0

TABLE 4.3: **Sensitivity Analysis.** Success rates (mean %) of various PERACT agents trained with 100 demonstrations per task. We investigate three factors that affect PERACT's performance: rotation augmentation, number of Perceiver latents, and voxel resolution.

with $\sim$ 33%, indicating that the global receptive field of the Transformer is crucial for solving the task.

4.4.4 Sensitivity Analysis

In Table 4.3, we investigate three factors that affect PERACT's performance: rotation data augmentation, number of Perceiver latents, and voxelization resolution. All multi-task agents were trained with 100 demonstrations per task and evaluated on 25 episodes per task. To briefly summarize these results: (1) 45° yaw perturbations improve performance on tasks with lots of rotation variations like `stack blocks`, but also worsen performance on tasks with constrained rotations like `place wine`. (2) PERACT with just 512 latents is competitive with (and sometimes even better than) the default agent with 2048 latents, which showcases the compression capability of the Perceiver architecture. (3) Coarse grids like 32^3 are sufficient for some tasks, but high-precision tasks like `sort shape` need higher resolution voxelization. (4) Large patch-sizes reduce memory usage, but they might affect tasks that need sub-patch precision.

4.4.5 High-Precision Tasks

PERACT achieves zero performance on three high-precision tasks: `place cups`, `stack cups`, and `insert peg`. To investigate if multi-task optimization is itself one of the factors affecting performance, we train 3 separate single-task agents for each task in Table 4.4. We find that single-task agents are able to achieve non-zero performance, indicating that better multi-task optimization methods might improve performance.

	Multi	Single
`place cups`	0	24
`stack cups`	0	32
`insert peg`	0	16

TABLE 4.4: Success rates (mean %) of multi-task and single-task PERACT agents trained with 100 demos and evaluated on 25 episodes.

4.4.6 Fine-Tuning on New Tasks

Pre-training foundation models [33] like CLIP [224] and BERT [69] offers the advantage of quickly fine-tuning on new tasks with minimal data – a technique that has become standard practice in vision and language. In Table 4.5, we present results for fine-tuning a PERACT agent on new manipulation tasks. We take the best pre-trained PERACT agent from Section 4.4.2,

	`take lid off`	`pickup cup`	`meat on grill`	`open bottle`	`pick and lift`	`remove cups`	`take roll`	`open oven`
Trained from scratch	88	16	48	76	20	0	28	12
Fine-tuned	96	44	48	84	48	12	32	8
Fine-tuned from Self-Attn	100	48	56	84	52	8	36	16
Fine-tuned from Decoder	100	44	72	84	32	24	56	12

TABLE 4.5: **Fine-Tuning PERACT on New Tasks.** Success rates (mean %) of various PERACT agents trained from scratch or fine-tuned on 8 new tasks, all with 10 demonstrations per task. Fine-tuning includes three variations: *(top)* all parameters, *(middle)* only parameters from self-attention layers onwards, *(bottom)* only parameters from decoder cross attention onwards, while the rest kept frozen.

which was trained on 18 tasks, and fine-tune it on 8 new RLBench [127] tasks with just 10 demonstrations per task. We compare against training PERACT from scratch on new tasks without any pre-training. Additionally, we present three types of fine-tuning: (1) updating all parameters in the agent, (2) only fine-tuning the parameters from the self-attention layers onwards, while keeping the rest frozen, and (3) only fine-tuning the parameters from the decoder cross-attention, while keeping the rest frozen.

These preliminary results indicate that pre-training PERACT improves its ability to learn new tasks. A pre-trained PERACT agent is already familiar with certain objects like cups and wine bottles, so it might be easier to acquire new skills that involve them. Furthermore, while some tasks like `open oven` involve completely unfamiliar objects like ovens, PERACT may have learnt some general 3D processing and reaching skills that make it more effective than starting from scratch. With larger models, fine-tuning and zero-shot capabilities can be further improved; PERACT with only 33M parameters is still tiny compared to the largest existing multimodal transformers like PaLM-E [74] with 562B parameters.

4.4.7 Real-Robot Results

We also validated our results with real-robot experiments on a Franka Emika Panda. Without any sim-to-real transfer or pre-training, we trained a multi-task PERACT agent *from scratch* on 7 tasks (with 18 unique variations) from a total of just 53 demonstrations. See the supplementary video for qualitative results that showcase the diversity of tasks and robustness to scene changes. Table 4.6 reports success rates from small-scale evaluations. Similar to the simulation results, we find that PERACT is able to achieve $>$ 65% success on simple short-horizon tasks like pressing hand-sanitizers from just a handful number of demonstrations. The most common failures involved predicting incorrect gripper open actions, which often lead the agent into unseen states. This could be addressed in future works by using HG-DAgger style approaches to correct the agent [129]. Other issues included the agent exploiting biases in the dataset like in prior work [250]. This could be addressed by scaling up expert data with more diverse tasks and task variants.

Task	# Train	# Test	Succ. %
Press Handsan	5	10	90
Put Marker	8	10	70
Place Food	8	10	60
Put in Drawer	8	10	40
Hit Ball	8	10	60
Stack Blocks	10	10	40
Sweep Beans	8	5	20

TABLE 4.6: Success rates (mean %) of a multi-task model trained an evaluated 7 real-world tasks.

4.4.8 Attention Visualization

Figure 4.9 presents selected examples of attention heatmaps from PERACT's encoder cross-attention. Notably, PERACT's receptive field effectively covers the entire input space of 100^3 voxels. For example, in the scenario where the task is to "use the stick to drag the cube onto the magenta target," the encoder attends to both the stick and the magenta target simultaneously. In contrast, some attention heatmaps are less interpretable. For the "open bottom drawer" example, the encoder mostly focuses on the floor. Such uninformative attention maps have been previously reported [248] with ViT-style transformers.

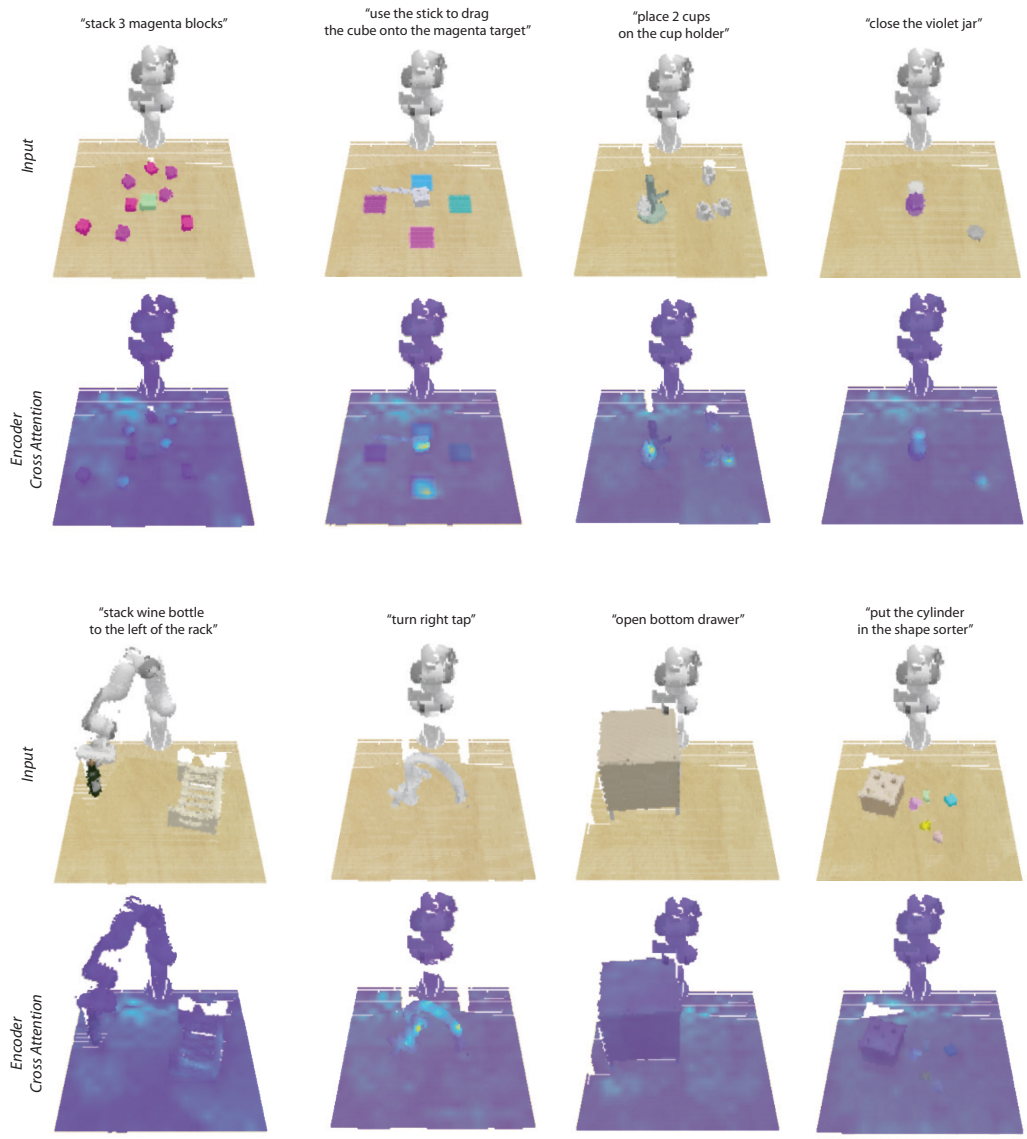


FIGURE 4.9: **Encoder Cross-Attention Visualization.** 8 examples of attention heatmaps from PERACT's encoder cross-attention that mixes input voxel patches with latent vectors.

4.5 Limitations and Risks

While PERACT is quite capable, it is not without limitations. In the following sections, we discuss some of these limitations and potential risks for real-world deployment.

Sampling-Based Motion Planner. PERACT relies on a sampling-based motion planner to execute discretized actions. This puts PERACT at the mercy of randomized planner to reach poses. While this issue did not cause any major problems with the tasks in our experiments, a lot of other tasks are sensitive to the paths taken to reach poses. For instance, pouring water into a cup would require a smooth path for tilting the water container appropriately. This could be

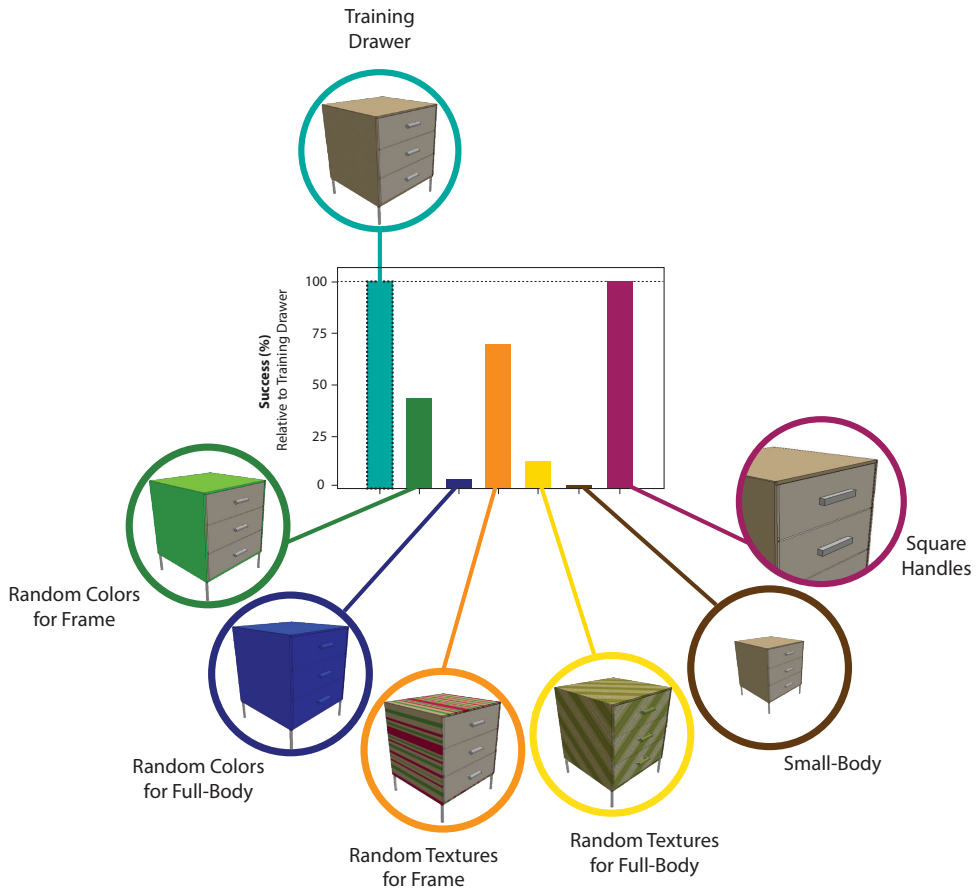


FIGURE 4.10: **Perturbation Tests.** Results from a multi-task PERACT agent trained on a single drawer and evaluated on several instances perturbed drawers. Each perturbation consists of 25 evaluation episodes, and reported successes are relative to the training drawer.

addressed in future works by using a combination of learned and sampled motion paths [123].

Dynamic Manipulation. Another issue with discrete-time discretized actions is that they are not easily applicable to dynamic tasks that require real-time closed-loop maneuvering. This could be addressed with a separate visuo-servoing mechanism that can reach target poses with closed-loop control. Alternatively, instead of predicting just one action, PERACT could be extended to predict a sequence of discretized actions. Here, the Transformer-based architecture could be particularly advantageous. Also, instead of just predicting poses, the agent could also be trained to predict other physical parameters like target velocities [312].

Dexterous Manipulation. Using discretized actions with N-DoF robots like multi-fingered hands is also non-trivial. Specifically for multi-fingered hands, PERACT could be modified to predict finger-tip poses that can be reached with an IK (Inverse Kinematics) solver. But it is unclear how feasible or robust such an approach would be with under-actuated systems like multi-fingered hands.

Generlization to Novel Instances and Objects. In Figure 4.10, we report results from small-scale perturbation experiments on the `open drawer` task. We observe that changing the shape of the handles does not affect performance. However, handles with randomized textures and colors confuse the agent since it has only seen one type of drawer color and texture during training. Going beyond this one-shot setting, and training on several instances of drawers might improve generalization performance. Although we did not explicitly study generalization to unseen objects, it might be feasible to train PERACT's action-detector on a broad range of objects and evaluate its ability to handle novel objects, akin to how language-conditioned instance-segmentors and object-detectors are used [142]. Alternatively, pre-trained vision features from multi-modal encoders like CLIP [224] or R3M [203] could be used to boostrap learning.

Scope of Language Grounding. Like with prior work [250], PERACT's understanding of verb-noun phrases is closely grounded in demonstrations and tasks. For example, "cleaning" in *"clean the beans on the table with a dustpan"* is specifically associated with the action sequence of pushing beans on to a dustpan, and not "cleaning" in general, which could be applied to other tasks like cleaning the table with a cloth.

Predicting Task Completion. For both real-world and simulated evaluations, an oracle indicates whether the desired goal has been reached. This oracle could be replaced with a success classifier that can be pre-trained to predict task completion from RGB-D observations.

History and Partial Observability. PERACT relies purely on the current observation to predict the next action. As such, tasks that require history like counting or ordering are not feasible, unless accompanied by a task-completion predictor. Similarly, for tasks involving partial observability e.g., looking through drawers one-by-one for a specific object, PERACT does not keep track of what was seen before. Future works could include observations from previous timesteps, or append Perceiver latents, or train a Recurrent Neural Network to encode latents across timesteps.

Data Augmentation with Kinematic Feasibility. The data augmentation method described in Section 4.3.4 does not consider the kinematic feasibility of reaching perturbed actions with the Franka arm. Future works could pre-compute unreachable poses in the discretized action space, and discard any augmentation perturbations that push actions into unreachable zones.

Balanced Datasets. Since PERACT is trained with just a few demonstrations, it occassionally tends to exploit biases in the training data. For instance, PERACT might have a tendency to always *"place blue blocks on yellow blocks"* if such an example is over-represented in the training data. Such issues could be potentially fixed by scaling datasets to include more diverse examples of objects and attributes. Additionally, data visualization methods could be used to identify and fix these biases.

Multi-Task Optimization. The uniform task sampling strategy presented in Section 4.3.4 might sometimes hurt performance. Since all tasks are weighted equally, optimizing for certain tasks with common elements (e.g., moving blocks), might adversarial affect the performance on other dissimilar tasks (e.g., turning taps). Future works, could use dynamic task-weighting methods like Auto-λ [164] for better multi-task optimization.

Deployment Risks. PERACT is an end-to-end framework for 6-DoF manipulation. Unlike some methods in Task-and-Motion-Planning that can sometimes provide theoretical guarantees on task completion, PERACT is a purely reactive system whose performance can only be evaluated through empirical means. Also, unlike prior works [250], we do not use internet pre-trained vision encoders that might contain harmful biases [25, 24]. Even so, it is prudent to thoroughly study and mitigate any biases before deployment. As such, for real-world applications, keeping humans in the loop both during training and testing, might help. Usage with unseen objects and observations with people is not recommended for safety critical systems.

4.6 Discussion

We presented PERACT, a Transformer-based multi-task agent for 6-DoF manipulation. Our experiments with both simulated and real-world tasks show that the right problem formulation, i.e., detecting voxel actions, makes a substantial difference in terms of data efficiency and robustness. Together with results from CLIPort, our experiments indicate that scaling-up robot learning with large-scale data is indeed possible. This paradigm is already well established in vision [107, 71] and NLP [69], so it is exciting to see where it takes the robot learning community next.

But both CLIPort and PERACT also suffer from a problem apparent in most vision and NLP systems: the inability to plan and reason across long-time horizons. Akin to most object-detectors in computer vision, our action-detectors are purely reactive: they take in an observation and output a single-step action at each time step. In partially-observable environments and tasks that involve hundreds or thousands time steps to solve, such reactive systems struggle to generalize and scale. One common way to deal with these issues is introduce abstractions, which is the topic of our next chapter.

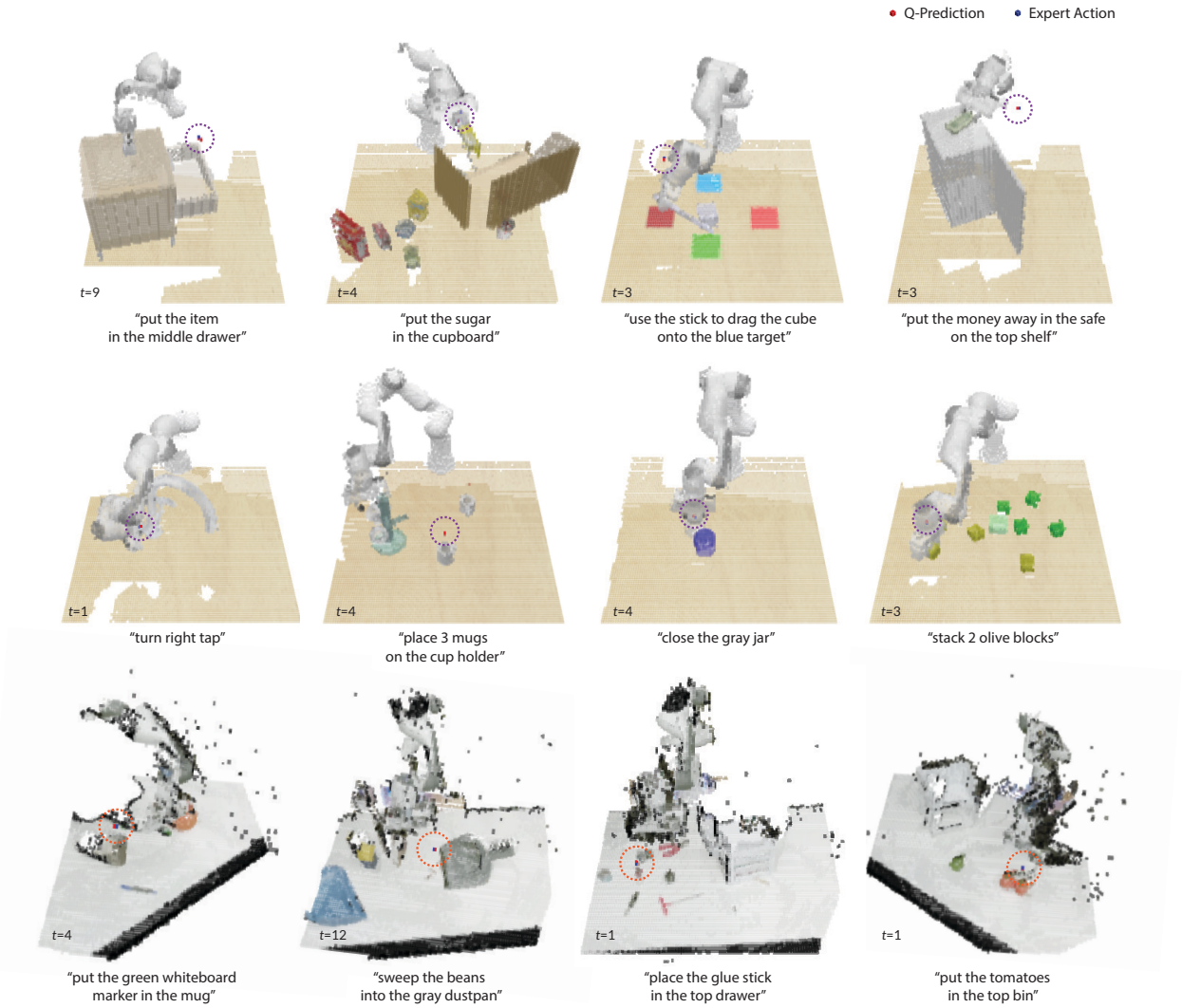


FIGURE 4.11: **Additional Q-Prediction Examples.** Translation Q-Prediction examples from PERACT. The top two rows are from simulated tasks without any data augmentation perturbations, and the bottom row is from real-world tasks with translation and yaw-rotation perturbations.

Chapter 5

Learning Abstract Policies in Language

Most useful goals take time to achieve. Both CLIPort and PERACT are limited to short-horizon tasks that take only a few seconds to complete like *"put the hammer in the top drawer"*. In contrast, real-life scenarios like *"making a cup of coffee"* or *"planning a trip to London"* could take minutes, days, or even months to complete.

One way humans deal with long time-horizons is by thinking purely in abstract terms like in language [21, 72]. By imagining action sequences and scoring their likelihood of success, prototypicality, and efficiency, humans can plan actions far into the future without moving a muscle. We then update our abstract plans as we receive concrete perceptual and physical feedback from our environments. Embodied agents require the same abilities, but existing work does not yet provide the infrastructure necessary for both reasoning abstractly and executing concretely.

In this chapter, we address this gap by introducing ALFWorld, a simulator that enables agents to learn abstract, language-based policies in TextWorld [57] (an interactive text game environment) and then execute goals from the ALFRED benchmark [252] in a rich visual environment. ALFWorld enables the creation of a new BUTLER agent whose abstract knowledge, learned in TextWorld, corresponds directly to concrete, visually grounded actions. In turn, as we demonstrate empirically, this fosters better agent generalization than training only in the visually grounded environment. BUTLER's simple, modular design factors the problem to allow researchers to focus on models for improving every piece of the pipeline (language understanding, planning, navigation, and visual scene understanding).

This chapter was published previously as Shridhar et al. [253]. Xingdi (Eric) Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht contributed to the materials presented here. The simulation environments, model code, data, and pre-trained models are available at alfworld.github.io.

5.1 Overview

FIGURE 5.1: **ALFWorld Environment.** Interactive aligned text and embodied worlds. An example with high-level text actions (left) and low-level physical actions (right).

Consider helping a friend prepare dinner in an unfamiliar house: when your friend asks you to clean and slice an apple for an appetizer, how would you approach the task? Intuitively, one could reason abstractly: (1) find an apple (2) wash the apple in the sink (3) put the clean apple on the cutting board (4) find a knife (5) use the knife to slice the apple (6) put the slices in a bowl. Even in an unfamiliar setting, abstract reasoning can help accomplish the goal by leveraging semantic priors. Priors like *locations of objects* – apples are commonly found in the kitchen along with implements for cleaning and slicing, *object affordances* – a sink is useful for washing an apple unlike a refrigerator, *preconditions* – better to wash an apple before slicing it, rather than the converse. We hypothesize that, learning to solve tasks using abstract language, unconstrained by the particulars of the physical world, enables agents to complete embodied tasks in novel environments by leveraging the kinds of semantic priors that are exposed by abstraction and interaction. To test this hypothesis, we have created the novel ALFWorld framework, the first interactive, parallel environment that aligns text descriptions and commands with physically embodied robotic simulation. We build ALFWorld by extending two prior works: TextWorld [57] - an engine for interactive text-based games, and ALFRED [252] - a large scale dataset for vision-language instruction following in embodied environments. ALFWorld provides two views of the same underlying world and two modes by which to interact with it: TextWorld, an abstract, text-based environment, generates textual observations of the world and responds to high-level text actions; ALFRED, the embodied simulator, renders the world in high-dimensional images and responds to low-level physical actions as from a robot (Figure 5.1).[1] Unlike prior work on instruction following [178, 13], which typically uses a static corpus of cross-modal expert demonstrations, we argue that aligned parallel environments like ALFWorld offer a distinct advantage: they allow agents to *explore, interact*, and *learn* in the abstract environment of language before encountering the complexities of the embodied environment.

While fields such as robotic control use simulators like MuJoCo [278] to provide infinite data through interaction, there has been no analogous mechanism – short of hiring a human

[1] *Note:* Throughout this work, for clarity of exposition, we use ALFRED to refer to both tasks and the grounded simulation environment, but rendering and physics are provided by AI2-THOR [148].

around the clock – for providing linguistic feedback and annotations to an embodied agent. TextWorld addresses this discrepancy by providing programmatic and aligned linguistic signals during agent exploration. This facilitates the first work, to our knowledge, in which an embodied agent learns the meaning of complex multi-step policies, expressed in language, directly through interaction.

Empowered by the ALFWorld framework, we introduce BUTLER (**B**uilding **U**nderstanding in **T**extworld via **L**anguage for **E**mbodied **R**easoning), an agent that first learns to perform abstract tasks in TextWorld using Imitation Learning (IL) and then transfers the learned policies to embodied tasks in ALFRED. When operating in the embodied world, BUTLER leverages the abstract understanding gained from TextWorld to generate text-based actions; these serve as high-level subgoals that facilitate physical action generation by a low-level controller. Broadly, we find that BUTLER is capable of generalizing in a zero-shot manner from TextWorld to unseen embodied tasks and settings. Our results show that training first in the abstract text-based environment is not only 7$\times$ faster, but also yields better performance than training from scratch in the embodied world. These results lend credibility to the hypothesis that solving abstract language-based tasks can help build priors that enable agents to generalize to unfamiliar embodied environments.

5.2 Related Work

The longstanding goal of grounding language learning in embodied settings [26] has lead to substantial work on interactive environments. ALFWorld extends that work with fully-interactive aligned environments that parallel textual interactions with photo-realistic renderings and physical interactions.

Interactive Text-Only Environments: We build on the work of text-based environments like TextWorld [57] and Jericho [103]. While these environment allow for textual interactions, they are not grounded in visual or physical modalities.

Vision and language: While substantial work exists on vision-language representation learning e.g., MAttNet [306], CMN [113], VQA [15], CLEVR [135], ViLBERT [171], they lack embodied or sequential decision making.

Embodied Language Learning: To address language learning in embodied domains, a number of interactive environments have been proposed: BabyAI [51], Room2Room [13], ALFRED [252], InteractiveQA [92], Das2018 [60], and NetHack [155]. These environments use language to communicate instructions, goals, or queries to the agent, but not as a fully-interactive textual modality.

Language for State and Action Representation: Others have used language for more than just goal-specification. Schwartz et al. [241] use language as an intermediate state to learn policies

in VizDoom. Similarly, Narasimhan et al. [204] and Zhong et al. [321] use language as an intermediate representation to transfer policies across different environments. Hu et al. [112] use a natural language instructor to command a low-level executor, and Jiang et al.[132] use language as an abstraction for hierarchical RL. However these works do not feature an interactive text environment for pre-training the agent in an abstract textual space. Zhu et al. [323] use high-level commands similar to ALFWorld to solve tasks in THOR with IL and RL-finetuning methods, but the policy only generalizes to a small set of tasks due to the vision-based state representation. Using symbolic representations for state and action is also an inherent characteristic of works in task-and-motion-planning [138, 150] and symbolic planning [19].

World Models: The concept of using TextWorld as a "game engine" to represent the world is broadly related to inverse graphics [154] and inverse dynamics [288] where abstract visual or physical models are used for reasoning and future predictions. Similarly, some results in cognitive science suggest that humans use language as a cheaper alternative to sensorimotor simulation [21, 72].

5.3 Aligning ALFRED and TextWorld

The ALFRED dataset [252], set in the AI2-THOR simulator [148], is a benchmark for learning to complete embodied household tasks using natural language instructions and egocentric visual observations. As shown in Figure 5.1 (right), ALFRED tasks pose challenging interaction and navigation problems to an agent in a high-fidelity simulated environment. Tasks are annotated with a goal description that describes the objective (e.g., "put a pan on the dining table"). We consider both template-based and human-annotated goals; further details on goal specification can be found in Appendix D.8. Agents observe the world through high-dimensional pixel images and interact using low-level action primitives: MoveAhead, RotateLeft/Right, LookUp/Down, Pickup, Put, Open, Close, and ToggleOn/Off.

Task type	# train	# seen	# unseen
Pick & Place	790	35	24
Examine in Light	308	13	18
Clean & Place	650	27	31
Heat & Place	459	16	23
Cool & Place	533	25	21
Pick Two & Place	813	24	17
All	3,553	140	134

TABLE 5.1: **Data Split.** Six ALFRED task types with heldout seen and unseen evaluation sets.

The ALFRED dataset also includes crowdsourced language instructions like "turn around and walk over to the microwave" that explain *how* to complete a goal in a step-by-step manner. We depart from the ALFRED challenge by omitting these step-by-step instructions and focusing on the more diffcult problem of using only on goal descriptions specifying *what* needs to be achieved.

Our aligned ALFWorld framework adopts six ALFRED `task-types` (Table 5.1) of various difficulty levels.[2] Tasks involve first *finding a particular object*, which often requires the agent to open and search receptacles like drawers or cabinets. Subsequently, all tasks other than Pick & Place require some interaction with the object such as heating (place object in microwave and

[2]To start with, we focus on a subset of the ALFRED dataset for training and evaluation that excludes tasks involving slicing objects or using portable container (e.g., bowls).

start it) or cleaning (wash object in a sink). To complete the task, the object must be placed in the designated location.

Within each task category there is significant variation: the embodied environment includes 120 `rooms` (30 kitchens, 30 bedrooms, 30 bathrooms, 30 living rooms), each dynamically populated with a set of portable `objects` (e.g., apple, mug), and static `receptacles` (e.g., microwave, fridge). For each task type we construct a larger train set, as well as seen and unseen validation evaluation sets:

(1) seen consists of known task instances {`task-type`, `object`, `receptacle`, `room`} in `rooms` seen during training, but with different instantiations of `object` locations, quantities, and visual appearances (e.g. two blue pencils on a shelf instead of three red pencils in a drawer seen in training).

(2) unseen consists of new task instances with possibly known `object-receptacle` pairs, but always in unseen `rooms` with different `receptacles` and scene layouts than in training tasks.

The seen set is designed to measure in-distribution generalization, whereas the unseen set measures out-of-distribution generalization. The scenes in ALFRED are visually diverse, so even the same task instance can lead to very distinct tasks, e.g., involving differently colored apples, shaped statues, or textured cabinets. For this reason, purely vision-based agents such as the unimodal baselines in Section 5.6.2 often struggle to generalize to unseen environments and objects.

The TextWorld framework [57] procedurally generates text-based environments for training and evaluating language-based agents. In order to extend TextWorld to create text-based analogs of each ALFRED scene, we adopt a common latent structure representing the state of the simulated world. ALFWorld uses PDDL - Planning Domain Definition Language [187] to describe each scene from ALFRED and to construct an equivalent text game using the TextWorld engine. The dynamics of each game are defined by the PDDL domain (see Appendix D.3 for additional details). Textual observations shown in Figure 5.1 are generated with templates sampled from a context-sensitive grammar designed for the ALFRED environments. For interaction, TextWorld environments use the following high-level actions:

`goto {recep}`	`take {obj} from {recep}`	`put {obj} in/on {recep}`
`open {recep}`	`close {recep}`	`toggle {obj}{recep}`
`clean {obj} with {recep}`	`heat {obj} with {recep}`	`cool {obj} with {recep}`

where `{obj}` and `{recep}` correspond to objects and receptacles. Note that `heat`, `cool`, `clean`, and `goto` are high-level actions that correspond to several low-level embodied actions.

ALFWorld, in summary, is an cross-modal framework featuring a diversity of embodied tasks with analogous text-based counterparts. Since both components are fully interactive, agents may be trained in either the language or embodied world and evaluated on heldout test tasks in either modality. We believe the equivalence between objects and interactions across modalities make ALFWorld an ideal framework for studying language grounding and cross-modal learning.

5.4 BUTLER: An Embodied Multi-Task Agent

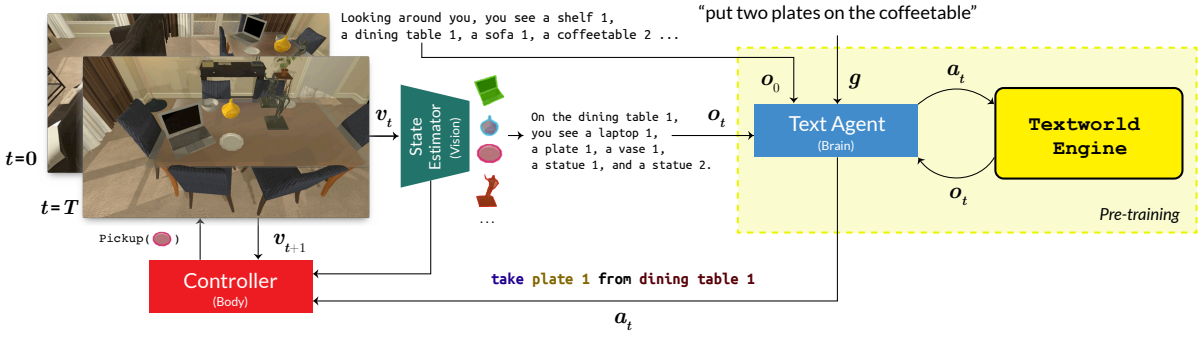


FIGURE 5.2: **BUTLER Agent** consists of three modular components. 1) BUTLER::BRAIN: a text agent pre-trained with the TextWorld engine (indicated by the dashed yellow box) which simulates an abstract textual equivalent of the embodied world. When subsequently applied to embodied tasks, it generates high-level actions that guide the controller. 2) BUTLER::VISION: a state estimator that translates, at each time step, the visual frame v_t from the embodied world into a textual observation o_t using a pre-trained Mask R-CNN detector. The generated observation o_t, the initial observation o_0, and the task goal g are used by the text agent the to predict the next high-level action a_t. 3) BUTLER::BODY: a controller that translates the high-level text action a_t into a sequence of one or more low-level embodied actions.

We investigate learning in the abstract language modality *before* generalizing to the embodied setting. The BUTLER agent uses three components to span the language and embodied modalities: BUTLER::BRAIN – the abstract text agent, BUTLER::VISION – the language state estimator, and BUTLER::BODY – the low-level controller. An overview of BUTLER is shown in Figure 5.2 and each component is described below.

5.4.1 BUTLER::BRAIN (Text Agent)

BUTLER::BRAIN is a novel text-based game agent that generates high-level text actions in a token-by-token fashion akin to Natural Language Generation (NLG) approaches for dialogue [246] and summarization [87]. An overview of the agent's architecture is shown in Figure 5.3. At game step t, the encoder takes the initial text observation o_0, current observation o_t, and the goal description g as input and generates a context-aware representation of the current state. The observation o_0 explicitly lists all the navigable receptacles in the scene, and goal g is sampled from a set of language templates (see Appendix D.8). Since the games are partially observable, the agent only has access to the observation describing the effects of its previous action and its present location. Therefore, we incorporate two memory mechanisms to imbue the agent with history: (1) a recurrent aggregator, adapted from [309], combines the encoded state with recurrent state h_{t-1} from the previous game step; (2) an observation queue feeds in the k most recent, unique

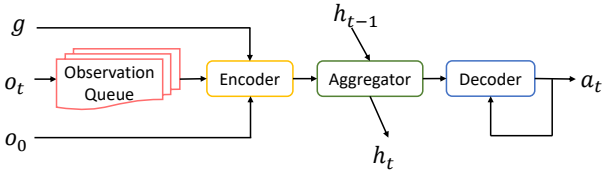


FIGURE 5.3: BUTLER::BRAIN. The text agent takes the initial/current observations o_0/o_t, and goal g to generate a textual action a_t token-by-token.

textual observations. The decoder generates an action sentence a_t token-by-token to interact with the game. The encoder and decoder are based on a Transformer Seq2Seq model with pointer softmax mechanism [96]. We leverage pre-trained BERT embeddings [237], and tie output embeddings with input embeddings [222]. The agent is trained in an imitation learning setting with DAgger [234] using expert demonstrations. See Appendix D.1 for complete details. When solving a task, an agent might get stuck at certain states due to various failures (e.g., action is grammatically incorrect, wrong object name). The observation for a failed action does not contain any useful feedback, so a fully deterministic actor tends to repeatedly produce the same incorrect action. To address this problem, during evaluation in both TextWorld and ALFRED, BUTLER::BRAIN uses Beam Search [227] to generate alternative action sentences in the event of a failed action. But otherwise greedily picks a sequence of best words for efficiency. Note that Beam Search is not used to optimize over embodied interactions like prior work [284]. but rather to simply improve the generated action sentence during failures.

5.4.2 BUTLER::VISION (State Estimator)

At test time, agents in the embodied world must operate purely from visual input. To this end, BUTLER::VISION's language state estimator functions as a captioning module that translates visual observations v_t into textual descriptions o_t. Specifically, we use a pre-trained Mask R-CNN detector [106] to identify objects in the visual frame. The detector is trained separately in a supervised setting with random frames from ALFRED training scenes (see Appendix D.4). For each frame v_t, the detector generates N detections $\{(c_1, m_1), (c_2, m_2), \ldots, (c_N, m_N)\}$, where c_n is the predicted object class, and m_n is a pixel-wise object mask. These detections are formatted into a sentence using a template e.g., `On table 1, you see a mug 1, a tomato 1, and a tomato 2`. To handle multiple instances of objects, each object is associated with a class c_n and a number ID e.g., `tomato 1`. Commands `goto`, `open`, and `examine` generate a list of detections, whereas all other commands generate affirmative responses if the action succeeds e.g., a_t: `put mug 1 on desk 2` $\rightarrow$ o_{t+1}: `You put mug 1 on desk 2`, otherwise produce `Nothing happens` to indicate failures or no state-change. See Appendix D.7 for a full list of templates. While this work presents preliminary results with template-based descriptions, future work could generate more descriptive observations using pre-trained image-captioning models [136], video-action captioning frameworks [262], or scene-graph parsers [267].

5.4.3 BUTLER::BODY (Controller)

The controller translates a high-level text action a_t into a sequence of L low-level physical actions $\{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_L\}$ that are executable in the embodied environment. The controller handles two types of commands: manipulation and navigation. For manipulation actions, we use the ALFRED API to interact with the simulator by providing an API action and a pixel-wise mask based on Mask R-CNN detections m_n that was produced during state-estimation. For navigation commands, each episode is initialized with a pre-built grid-map of the scene, where each receptacle instance is associated with a receptacle class and an interaction viewpoint (x, y, θ, ϕ) with x and y representing the 2D position, θ and ϕ representing the agent's yaw rotation and

camera tilt. The `goto` command invokes an A* planner to find the shortest path between two viewpoints. The planner outputs a sequence of L displacements in terms of motion primitives: MOVEAHEAD, ROTATERIGHT, ROTATELEFT, LOOKUP, and LOOKDOWN, which are executed in an open-loop fashion via the ALFRED API. We note that a given pre-built grid-map of receptacle locations is a strong prior assumption, but future work could incorporate existing models from the vision-language navigation literature [13, 284] for map-free navigation.

5.5 Experiments

We design experiments to answer the following questions: (1) How important is an *interactive* language environment versus a static corpus? (2) Do policies learnt in TextWorld transfer to embodied environments? (3) Can policies generalize to human-annotated goals? (4) Does pre-training in an abstract textual environment enable better generalization in the embodied world?

5.5.1 Importance of Interactive Language

The first question addresses our core hypothesis that training agents in interactive TextWorld environments leads to better generalization than training agents with a static linguistic corpus. To test this hypothesis, we use DAgger [234] to train the BUTLER::BRAIN agent in TextWorld and compare it against **Seq2Seq**, an identical agent trained with Behavior Cloning from an equivalently-sized corpus of expert demonstrations. The demonstrations come from the same expert policies and we control the number of episodes to ensure a fair comparison. Table 5.2 presents results for agents trained in TextWorld and subsequently evaluated in embodied environments in a zero-shot manner. The agents are trained independently on individual tasks and also jointly on all six task types. For each task category, we select the agent with best evaluation performance in TextWorld (from 8 random seeds); this is done separately for each split: seen and unseen. These best-performing agents are then evaluated on the heldout seen and unseen embodied ALFREDtasks. For embodied evaluations, we also report goal-condition success rates, a metric proposed in ALFRED [252] to measure partial goal completion.[3]

Comparing **ALFWorld** to **Seq2Seq**, we see improved performance on all types of seen tasks and five of the seven types of unseen tasks, supporting the hypothesis that interactive TextWorld training is a key component in generalizing to unseen embodied tasks. Interactive language not only allows agents to explore and build an understanding of successful action patterns, but also to recover from mistakes. Through trial-and-error the BUTLER agent learns task-guided heuristics, e.g., searching all the drawers in kitchen to look for a knife. As Table 5.2 shows, these heuristics are subsequently more capable of generalizing to the embodied world. More details on TextWorld training and generalization performance can be found in Section 5.6.1.

[3]For instance, the task "put a hot potato on the countertop" is composed of three goal-conditions: (1) heating some object, (2) putting a potato on the countertop, (3) heating a potato and putting it on the countertop. If the agent manages to put any potato on the countertop, then $1/3 = 0.33$ goal-conditions are satisfied, and so on.

task-type	TextWorld seen	TextWorld unseen	Seq2Seq seen	Seq2Seq unseen	ALFWorld seen	ALFWorld unseen	ALFWorld-ORACLE seen	ALFWorld-ORACLE unseen	Human Goals seen	Human Goals unseen
Pick & Place	69	50	28 (28)	17 (17)	**30 (30)**	**24 (24)**	53 (53)	31 (31)	20 (20)	10 (10)
Examine in Light	69	39	5 (13)	0 (6)	**10 (26)**	0 **(15)**	22 (41)	12 (37)	2 (9)	0 (8)
Clean & Place	67	74	**32** (41)	12 (31)	**32 (46)**	**22 (39)**	44 (57)	41 (56)	18 (31)	22 (39)
Heat & Place	88	83	10 (29)	12 (33)	**17 (38)**	**16 (39)**	60 (66)	60 (72)	8 (29)	5 (30)
Cool & Place	76	91	2 (19)	**21 (34)**	**5 (21)**	19 (33)	41 (49)	27 (44)	7 (26)	17 (34)
Pick Two & Place	54	65	12 (23)	0 (26)	**15 (33)**	**8 (30)**	32 (42)	29 (44)	6 (16)	0 (6)
All Tasks	40	35	6 (15)	5 (14)	**19 (31)**	**10 (20)**	37 (46)	26 (37)	8 (17)	3 (12)

TABLE 5.2: **Zero-shot Domain Transfer**. *Left*: Success percentages of the best BUTLER::BRAIN agents evaluated purely in TextWorld. *Mid-Left*: Success percentages after zero-shot transfer to embodied environments. *Mid-Right*: Success percentages of BUTLER with an oracle state-estimator and controller, an upper-bound. *Right*: Success percentages of BUTLER with human-annotated goal descriptions, an additional source of generalization difficulty. All successes are averaged across three evaluation runs. Goal-condition success rates [252] are given in parentheses. The **Seq2Seq** baseline is trained in TextWorld from pre-recorded expert demonstrations using standard supervised learning. **ALFWorld** is our main model using the Mask R-CNN detector and A* navigator. **ALFWorld-ORACLE** uses an oracle state-estimator with ground-truth object detections and an oracle controller that directly teleports between locations.

5.5.2 Transferring to Embodied Tasks

Since TextWorld is an abstraction of the embodied world, transferring between modalities involves overcoming *domain gaps* that are present in the real world but not in TextWorld. For example, the physical size of objects and receptacles must be respected – while TextWorld will allow certain objects to be placed inside any receptacle, in the embodied world it might be impossible to put a larger object into a small receptacle (e.g. a large pot into a microwave).

Subsequently, a TextWorld-trained agent's ability to solve embodied tasks is hindered by these domain gaps. So to study the transferability of the text agent in isolation, we introduce **ALFWorld-ORACLE** in Table 5.2, an oracle variant of BUTLER which uses perfect state-estimation, object-detection, and navigation. Despite these advantages, we nevertheless observe a notable drop in performance from **TextWorld** to **ALFWorld-ORACLE**. This performance gap results from the domain gaps described above as well as misdetections from Mask R-CNN and navigation failures caused by collisions. Future work might address this issue by reducing the domain gap between the two environments, or performing additional fine-tuning in the embodied setting.

The supplementary video contains qualitative examples of the BUTLER agent solving tasks in unseen environments. It showcases 3 successes and 1 failure of a TextWorld-only agent trained on All Tasks. In "put a watch in the safe", the agent has never seen the 'watch'-'safe' combination as a goal.

5.5.3 Generalizing to Human-Annotated Goals

BUTLER is trained with templated language, but in realistic scenarios, goals are often posed with open-ended natural language. In Table 5.2, we present **Human Goals** results of BUTLER evaluated on human-annotated ALFREDgoals, which contain 66 unseen verbs (e.g., 'wash', 'grab', 'chill') and 189 unseen nouns (e.g., 'rag', 'lotion', 'disc'; see Appendix D.8 for full list).

Surprisingly, we find non-trivial goal-completion rate indicating that certain categories of task, such as pick and place, are quite generalizable to human language. While these preliminary results with natural language are encouraging, we expect future work could augment the templated language with synthetic-to-real transfer methods [184] for better generalization.

5.5.4 To Pretrain or not to Pretrain in TextWorld?

Given the domain gap between TextWorld and the embodied world, *Why not eliminate this gap by training from scratch in the embodied world?* To answer this question, we investigate three training strategies: (i) EMBODIED-ONLY: pure embodied training, (ii) TW-ONLY: pure TextWorld training followed by zero-shot embodied transfer and (iii) HYBRID training that switches between the two environments with 75% probability for TextWorld and 25% for embodied world. Table 5.3 presents success rates for these agents trained and evaluated on All Tasks. All evaluations were conducted with an oracle state-estimator and controller. For a fair comparison, each agent is trained for 50K episodes and the training speed is recorded for each strategy. We report peak performance for each split.

Training Strategy	train succ %	seen succ %	unseen succ %	train speed eps/s
EMBODIED-ONLY	21.6	**33.6**	23.1	0.9
TW-ONLY	23.1	27.1	**34.3**	**6.1**
HYBRID	11.9	21.4	23.1	0.7

TABLE 5.3: **Training Strategy Success**. Trained on All Tasks for 50K episodes and evaluated in embodied scenes using an oracle state-estimator and controller.

Results indicate that TW-ONLY generalizes better to unseen environments while EMBODIED-ONLY quickly overfits to seen environments (even with a perfect object detector and teleport navigator). We hypothesize that the abstract TextWorld environment allows the agent to focus on quickly learning tasks without having to deal execution-failures and expert-failures caused by physical constraints inherent to embodied environments. TextWorld training is also 7× faster[4] since it does not require running a rendering or physics engine like in the embodied setting. See Section D.6 for more quantitative evaluations on the benefits of training in TextWorld.

5.6 Ablations

We conduct ablation studies to further investigate: (1) The generalization performance of BUTLER::BRAIN within TextWorld environments, (2) The ability of unimodal agents to learn directly through visual observations or action history, (3) The importance of various hyper-parameters and modeling choices for the performance of BUTLER::BRAIN.

[4]For a fair comparison, all agents in Table 5.3 use a batch-size of 10. AI2-THOR instances use 100MB×batch-size of GPU memory for rendering, whereas TextWorld instances are CPU-only and are thus much easier to scale up.

	Pick & Place			Examine in Light			Clean & Place			Heat & Place			Cool & Place			Pick Two & Place			All Tasks		
	tn	sn	un	tn	sn	un	tn	sn	un	tn	sn	un	tn	sn	un	tn	sn	un	tn	sn	un
BUTLER	*54*	**61**	**46**	*59*	**39**	**22**	*37*	**44**	39	*60*	**81**	**74**	*46*	**60**	**100**	*27*	29	**24**	*16*	**40**	**37**
BUTLER$_g$	*54*	43	33	*59*	31	17	*37*	30	26	*60*	69	70	*46*	50	76	*27*	**38**	12	*16*	19	22
Seq2Seq	*31*	26	8	*44*	31	11	*34*	30	**42**	*36*	50	30	*27*	32	33	*17*	8	6	*9*	10	9

TABLE 5.4: **Generalization within TextWorld environments.** We independently train BUTLER::BRAIN on each type of TextWorld task and evaluate on heldout scenes of the same type. Respectively, tn/sn/un indicate success rate on train/seen/unseen tasks. All sn and un scores are computed using the random seeds (from 8 in total) producing the best final training score on each task type. BUTLER is trained with DAgger and performs beam search during evaluation. Without beam search, BUTLER$_g$ decodes actions *greedily* and gets stuck repeating failed actions. Further removing DAgger and training the model in a Seq2Seq fashion leads to worse generalization. Note that tn scores for BUTLER are lower than sn and un as they were computed without beam search.

5.6.1 Generalization within TextWorld

We train and evaluate BUTLER::BRAIN in abstract TextWorld environments spanning the six tasks in Table 5.4, as well as All Tasks. Similar to the zero-shot results presented in Section 5.5.2, the All Tasks setting shows the extent to which a *single* policy can learn and generalize on the large set of 3,553 different tasks, but here without having to deal with failures from embodied execution.

We first experimented with training BUTLER::BRAIN through reinforcement learning (RL) where the agent is rewarded after completing a goal. Due to the infesibility of using candidate commands or command templates as discussed in Section D.9, the RL agent had to generate actions token-by-token. Since the probability of randomly stumbling upon a grammatically correct and contextually valid action is very low (7.02e-44 for sequence length 10), the RL agent struggled to make any meaningful progress towards the tasks.

After concluding that current reinforcement learning approaches were not successful on our set of training tasks, we turned to DAgger [234] assisted by a rule-based expert (detailed in Appendix D.5). BUTLER::BRAIN is trained for 100K episodes using data collected by interacting with the set of training games.

Results in Table 5.4 show (i) Training success rate varies from 16-60% depending on the category of tasks, illustrating the challenge of solving hundreds to thousands of training tasks within each category. (ii) Transferring from training to heldout test games typically reduces performance, with the unseen rooms leading to the largest performance drops. Notable exceptions include heat and cool tasks where unseen performance exceeds training performance. (iii) Beam search is a key contributor to test performance; its ablation causes a performance drop of 21% on the seen split of All Tasks. (iv) Further ablating the DAgger strategy and directly training a Sequence-to-Sequence (Seq2Seq) model with pre-recorded expert demonstrations causes a bigger performance drop of 30% on seen split of All Tasks. These results suggest that online interaction with the environment, as facilitated by DAgger learning and beam search, is essential for recovering from mistakes and sub-optimal behavior.

5.6.2 Unimodal Baselines

Agent	seen (succ %)	unseen (succ %)
BUTLER	**18.8**	**10.1**
VISION (RESNET18)	10.0	6.0
VISION (MCNN-FPN)	11.4	4.5
ACTION-ONLY	0.0	0.0

TABLE 5.5: **Unimodal Baselines**. Trained on All Tasks with 50K episodes and evaluated in the embodied environment.

Table 5.5 presents results for unimodal baseline comparisons to BUTLER. For all baselines, the action space and controller are fixed, but the state space is substituted with different modalities. To study the agents' capability of learning a single policy that generalizes across various tasks, we train and evaluate on All Tasks. In VISION (RESNET18), the textual observation from the state-estimator is replaced with ResNet-18 `fc7` features [107] from the visual frame. Similarly, VISION (MCNN-FPN) uses the pre-trained Mask R-CNN from the state-estimator to extract FPN layer features for the whole image. ACTION-ONLY acts without any visual or textual feedback. We report peak performance for each split.

The visual models tend to overfit to seen environments and generalize poorly to unfamiliar environments. Operating in text-space allows better transfer of policies without needing to learn state representations that are robust to visually diverse environments. The zero-performing ACTION-ONLY baseline indicates that memorizing action sequences is an infeasible strategy for agents.

5.6.3 Model Ablations

FIGURE 5.4: Model ablations on All Tasks. x-axis: 0 to 50k episodes; y-axis: normalized success from 0 to 75%.

Figure 5.4 illustrates more factors that affect the performance of BUTLER::BRAIN. The three rows of plots show training curves, evaluation curves in seen and unseen settings, respectively. All experiments were trained and evaluated on All Tasks with 8 random seeds.

In the first column, we show the effect of using different observation queue lengths k as described in Section 5.4.1, in which size 0 refers to not providing any observation information to the agent. In the second column, we examine the effect of explicitly keeping the initial observation o_0, which lists all the receptacles in the scene. Keeping the initial observation o_0 facilitates the decoder to generate receptacle words more accurately for unseen tasks, but may be unnecessary in seen environments. The third column suggests that the recurrent component in our aggregator is helpful in making history-based decisions particularly in seen environments where keeping track of object locations is useful. Finally, in the fourth column, we see that using more training games can lead to better generalizability in both seen and unseen settings. Fewer training games achieve high training scores by quickly overfitting, which lead to zero evaluation scores.

5.7 Limitations

Although BUTLER's design nicely abstracts away the complexities of embodied environments, it somewhat contradicts the insights from CLIPort and PERACT that argue for end-to-end approaches without intermediate states. This leads to some familiar issues from traditional robotics.

Dependence on Good State-Estimator and Low-Level Controller. BUTLER's success relies on a good state-estimator and low-level controller. Any errors in language descriptions generated by the state-estimator or failures in the open-loop execution of high-level actions is hard to correct with a text-only controller. While the use of abstractions to decouple the planning process from perception and control harks back approaches in traditional symbolic planning [187, 138], it is unclear if planning can be entirely decoupled from sensing and actuation.

Using Language as the State Representation. Using language as the state representation is intuitive and appealing, but not everything can be described in language. Consider the long and elaborate process of making hand-pulled noodles[5]. Language might be great for describing high-level details like *the dough needs to be stretched*, but no so much for low-level things like *how exactly each piece of dough should be stretched*. Perhaps language can only be used to capture high-level abstractions of long-horizon processes, but it is unclear when and where exactly this boundary of abstraction begins and ends. Also, it might be better to acquire or evolve language through natural interactions rather than rely on human annotations.

Sim-to-Real Transfer. Similar to ALFRED, the low fidelity of both TextWorld and embodied simulators might limit sim-to-real transfer capabilities of pre-trained agents. The TextWorld Engine is limited in scope in terms of linguistic diversity, and AI2-THOR abstracts away real-world issues like sensor noise and non-deterministic actions. Nonetheless, experimental results from ALFWorld indicate that approaches like BUTLER might scale effectively in the real-world given sufficient data.

Use of Object Detectors. The use of Mask-RCNN for detecting objects for state-estimation is also limiting. Some objects might be too small to "detect" like flour particles, and some may not be amenable to the concept of *objects* at all, such as water, steam, noodles, and cloths. The bottleneck of using detector outputs also suffers from similar limitations to using language as the state representation: not everything can be "detected" easily.

5.8 Discussion

We introduced ALFWorld, the first interactive text environment with aligned embodied worlds. ALFWorld allows agents to explore, interact, and learn abstract polices in a textual environment. Pre-training our novel BUTLER agent in TextWorld, we show zero-shot generalization to embodied tasks in the ALFRED dataset. The results indicate that reasoning in textual space allows for better generalization to unseen tasks and also faster training, compared to other modalities like vision.

[5] https://www.youtube.com/watch?v=f2kesmA08VU

Chapter 6

Counterpoints

Like with most conceptual frameworks, ideas are not without flaws. In this section, we present some counterpoints to the arguments and formulations made in this thesis. We briefly describe scenarios in which language, action-centric representations, end-to-end learning, and embodiment might be limited in solving certain problems or hinder generalization capabilities.

6.1 Language is limited for goal-specification and planning

Language is an intuitive medium for specifying goals, but sometimes it is tedious or infeasible to describe all the details. Consider the task of *"baking a rainbow cake"*. There are thousands of possible "rainbow cakes" that fit the criteria, but a user might have a specific type of cake in mind, with certain frosting, toppings, and decorations. It is much easier to show a photo of a rainbow cake than to communicate these details in language.

Similarly, a symbolic medium like language might not be strictly necessary for long-horizon planning. Natural language is at most 150,000 years old, but humans have been capable of carrying out very long-horizon tasks far before this period. Furthermore, other animals like cats and dogs are considerably more capable than today's robots in terms of physical skills, despite lacking human-like languages. It is unclear if language is a wrapper around long-term and abstract reasoning, or if language itself is part of the reasoning mechanism (although some recent evidence [76] supports the former case).

6.2 Action-centric agents struggle with compositional generalization

While CLIPort and PERACT can manipulate unconventional objects like deformable materials and granular media, their action-centric representations struggle to exhibit the same compositional generalization as traditional object-centric representations. This is apparent in our simulation results (Section 3.4.2), where we observe that agents are not perfect at putting together seen concepts in unseen goal compositions. Object-centric representations represent scenes with reusable symbolic units that naturally allow for novel compositions. By abstracting away low-level details on what makes a mug *a mug*, object-centric models avoid over-fitting representations to specific tasks and environments, e.g., the exact texture of a mug or its surrounding objects. In scenarios where compositional grounding has to be guaranteed either due to

safety or high-performance requirements, traditional object-centric representations might be still preferable. For instance, a self-driving car reading a road-sign might be safer running an object-detector to discern direction arrows, instead of determining the sign direction in an end-to-end image-to-action fashion.

6.3 Just behavior-cloning is insufficient

ALFRED, CLIPort, PERACT, and ALFWorld are all trained with behavior-cloning. However, this supervised learning paradigm is limited in generalizing beyond the training data. For vision and language models, internet data might sufficiently cover a broad range of scenarios in their respective domains, but this is not the case for robotics. It is difficult, if not impossible, to account for all possible scenarios across different environments, timescales, and state-action spaces. Furthermore, behavior-cloning assumes that the data used to train the agent is collected by an expert, typically a human user. Asking a human to demonstrate what the robot should do in all possible scenarios is physically infeasible. These limitations hinder the scalability and generalization capabilities of trained agents.

To address these issues, one alternative paradigm is to use reinforcement learning for both data collection and exploration of novel environments. However, this approach presents additional challenges, such as the need for a reward model, reset mechanism, and large amounts of environment interactions. Recent methods have attempted to overcome these challenges by fine-tuning behavior-cloned policies with reinforcement learning [157]. By doing so, agents can learn to explore new environments and generalize beyond the training data.

6.4 Purely reactive systems are limited

For a lot of tasks, reactive systems like CLIPort and PERACT are insufficient. Reactive systems map observations directly to actions without considering past experiences or anticipating future events. Partially-observable tasks require the ability to remember past experiences, such as retrieving items from a previously checked drawer. In humans, memory plays a significant role in decision-making. Different types of memory, such as muscle memory for motor skills, semantic memory for general knowledge, and episodic memory for events, are crucial for solving daily tasks. Without memory, agents can become stuck in loops and fail to make progress.

In addition to memory, foresight is another essential factor for making decisions. Humans use foresight to plan actions in advance and avoid suboptimal behavior in the long-run. While ALFWorld briefly explores learning high-level plans in language, it does not choose between potential future actions based on some forward model of the world. Overall, relying solely on reactive methods does not provide the level of decision-making required for complex tasks that require memory and foresight.

6.5 Embodiment is not strictly necessary

While this thesis argues for the importance of grounding vision and language in embodiment and actions, recent disembodied models such as GPT [37] and CLIP [224] have demonstrated impressive capabilities in understanding the physical world. These models suggest that it is possible to learn some representations of language using language-only data, and of vision using mostly visual data. For tasks like Optical Character Recognition (OCR) that involve reading text in images, or answering questions about a document, or engaging in abstract conversations, embodiment is not strictly necessary.

Further, the latest generative models like ChatGPT and GPT-4 demonstrate a decent understanding of physical commonsense [38] despite not being trained on embodied data. It may be that the internet has good enough converge of physical phenomenon in text format. However, it is unclear if this text-based information is comprehensive enough to cover future events that we humans have yet to experience, such as the introduction of new gadgets and our subsequent physical interactions with them. It remains to be seen if disembodied models possess the ability to reason about these future events with the same level of accuracy and efficiency as embodied agents.

6.6 End-to-end robot learning is limited

Although end-to-end learning is an appealing paradigm, it has a few drawbacks. It can be difficult to understand why a particular action was taken and how to correct undesired behavior. This lack of transparency could be problematic if end-to-end learning systems are deployed in real-world environments with minimal human supervision, potentially resulting in safety issues. Additionally, end-to-end learning models are susceptible to adversarial attacks, whereby small perturbations to the input could dramatically alter the agent's output [75].

Furthermore, the end-to-end learning paradigm lacks modularity, which can limit the re-usability of individual components of the system. This limited re-usability also results in increased data requirements where end-to-end methods usually rely on significantly more data than modular counterparts.

Chapter 7

Conclusion

In this thesis, we presented various methods for embodied agents to *act with language* by integrating paradigms from computer vision, natural language processing, and robotics. First, we presented a large-scale benchmark for guiding agents with language goals in simulated household environments. This problem setup highlighted several challenges for future butler robots, such as long-horizon reasoning, partial observability, and irrerversible actions. Next, we presented methods to enhance the expressivity of the agent's action-space by directly grounding language goals in precise 2D and 3D manipulation actions. By aligning the perception and action spaces, we learned *perceptual representations of actions conditioned on language goals* in a data-efficient manner. Further, by avoiding explicit scene representations like symbols, graphs, object poses, and segmentations, we could directly predicted actions for manipulating deformable objects, granular media, and other unconventional objects. Finally, we extended these capabilities to long-horizon tasks by using language as a medium for learning abstract policies.

All these approaches showcase the power of scaling-up simple methods with data to solve complex robotics problems. But as with any research area, a number of open challenges remain. In the following sections, we highlight some potential future directions.

Dynamic and Continuous Tasks. One interesting direction is extending end-to-end manipulation to highly dynamic tasks like throwing, twisting, pouring, and forceful interactions. Currently, CLIPort and PERACT use a discretized action-space that is applicable to a wide range of quasi-static tasks. However, extending this approach to non-quasi-static tasks that involve high-frequency perception-action feedback is non-trivial. One simple extension is to predict a sequence of future actions for k steps and execute only the first $k - t$ actions, following a Model-Predictive-Control (MPC) paradigm. Also, instead of selecting a single pixel or voxel location, the model could be trained to predict entire 2D/3D trajectories or waypoints, and use splining to generate smooth continuous actions. Dynamic behavior may also require the prediction of other physical parameters like target velocities, accelerations, and jerks, in addition to end-effector positions. This approach has been successful in TossingBot [312], which predicts target velocities for throwing objects in addition to grasping affordances.

More Embodiments. Current robotics hardware is not limited to single-arm manipulators. There exist other embodiments, such as bi-manual manipulators, multi-fingered hands, and suction-and-parallel gripper hybrids, that can be easily mounted on mobile platforms, including holonomic and non-holonomic wheel bases, legged robots, aerial vehicles, and underwater vehicles. While extending PERACT and CLIPort to high-dimensional action-spaces, such as those of multi-fingered hands, is generally challenging, it might be possible to adapt them for other embodiments by aligning the observation and action spaces. For bi-manual manipulation, PERACT's decoder could be split to predict two gripper actions. For suction-and-parallel gripper hybrids, separate affordance maps could be predicted for each modality [313]. For navigation with wheeled robots on flat terrains, CLIPort could predict 2D waypoints on a birds-eye map. Similarly, for navigating in airspace or underwater, PERACT could predict 3D waypoints and trajectories.

Scaling-up Data Collection. The biggest bottleneck in scaling-up robot learning is collecting data. Unlike scraping images or text from the internet, teleoping (or tele-operating) demonstrations with physical robots is time consuming, expensive, and hard to parallelize. One solution could be to build better teleop systems with low cost hardware [318] to speed-up and improve the ease of data collection. Using better interfaces like virtual-reality setups [18] and hand-tracking based controllers [2] could also help. Alternatively, learning skills from human videos on the internet [295, 247] could reduce the amount of *in situ* data that has to be physically collected with a robot. Beside internet videos, it might be feasible to collect large amounts of "play data" by allowing humans to explore the scene quickly [283] without being slowed down by teleop.

Better Integration with Internet-Pretrained Models. Despite efforts to scale-up robot data, it is unlikely that they can match the sheer quantity and diversity of data available on the internet. Therefore, making use of pre-trained vision and language models from the internet to bootstrap learning will continue to be important for robot learning, at least in the near future. While CLIPort was a first attempt in using pre-trained vision-language models like CLIP, there might be better ways of integrating such models. In Flamingo [9], a frozen language model (Chinchilla-70B [110]) is surgically spliced between vision encoders to train a general-purpose vision-model. Since the language model is trained on significantly more text data, keeping it frozen avoids catastrophic forgetting when fine-tuned on much smaller vision datasets. And since there is even less data in robotics, it makes sense to freeze both the vision and language layers of a large model like Flamingo, and only fine-tune some action-decoder layers with robot data. This approach could potentially lead to some new zero-shot generalization capabilities that robots currently lack. However, it is not easy to fine-tune large models like Chinchilla or Flamingo in practice[1].

[1] https://docs.google.com/document/d/1ZNGyVWYFUbzV0xuei4SED2QAakGjMpaaQALcKYQm46U/edit

Safety and Ethical Considerations. The use of internet pre-trained models for robotics applications raises a number of safety and ethical concerns. Large language models and vision-language frameworks are prone to biases on the internet [25, 24, 90] that contain harmful stereotypes related to gender, race, and sexual orientation. Directly using these frameworks in end-to-end robot learning applications could lead to unintended consequences that might harm certain groups of people. Unlike passive applications like image or text generation, these issues are further exacerbated by the fact that robots can take physical actions in the real-world. It is prudent to thoroughly study and mitigate any biases before deploying pre-trained agents. Therefore, for real-world applications, it is recommended to have human supervisors in the loop during both training and testing. The usage of internet pre-trained models with unseen objects and observations with humans is not recommended for safety-critical systems.

***

Language provides a natural interface for guiding agents to achieve goals, and also a powerful tool for abstracting complex perceptual and physical phenomenon with generalizable representations. Integrating vision and language with agency brings us one-step closer to the long-standing dream of assistive robots that can take care of our everyday needs and alleviate the burden of human labor.

Bibliography

[1] Google scanned objects dataset, 2020.

[2] From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation, 2022.

[3] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L. Hamilton. Learning dynamic belief graphs to generalize on text-based games. In *NeurIPS*, 2020.

[4] Shubham Agrawal, Yulong Li, Jen-Shuo Liu, Steven K Feiner, and Shuran Song. Scene editing as teleoperation: A case study in 6dof kit assembly. *arXiv:2110.04450*, 2021.

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

[6] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv:1910.07113*, 2019.

[7] Huda Alamri, Vincent Cartillier, Abhishek Das, Jue Wang, Anoop Cherian, Irfan Essa, Dhruv Batra, Tim K. Marks, Chiori Hori, Peter Anderson, Stefan Lee, and Devi Parikh. Audio-visual scene-aware dialog. In *CVPR*, 2019.

[8] Jean-Baptiste Alayrac, Piotr Bojanowski, Nishant Agrawal, Ivan Laptev, Josef Sivic, and Simon Lacoste-Julien. Unsupervised learning from narrated instruction videos. In *CVPR*, 2016.

[9] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv:2204.14198*, 2022.

[10] Prithviraj Ammanabrolu and Matthew Hausknecht. Graph constrained reinforcement learning for natural language action spaces. In *ICLR*, 2020.

[11] Michael L Anderson. Embodied cognition: A field guide. *Artificial Intelligence*, 149(1):91–130, 2003.

[12] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018.

[13] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.

[14] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, June 2016.

[15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *ICCV*, 2015.

[16] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 2013.

[17] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Edward C. Williams, Mina Rhee, Lawson L. Wong, and Stefanie Tellex. Grounding natural language instructions to semantic goal representations for abstraction and generalization. *AuRo*, 2019.

[18] Sridhar Pandian Arunachalam, Irmak Güzey, Soumith Chintala, and Lerrel Pinto. Holodex: Teaching dexterity with immersive mixed reality. *arXiv:2210.06463*, 2022.

[19] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*, 2018.

[20] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[21] Briony Banks, Cai Wingfield, and Louise Connell. Linguistic distributional knowledge and sensorimotor grounding both contribute to semantic category production. 2020.

[22] Roger G Barker. Explorations in ecological psychology. *American Psychologist*, 20(1):1, 1965.

[23] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic roommates making pancakes. In *IEEE-RAS*, 2011.

[24] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *FAccT*, 2021.

[25] Abeba Birhane, Vinay Uday Prabhu, and Emmanuel Kahembwe. Multimodal datasets: misogyny, pornography, and malignant stereotypes. *arXiv:2110.01963*, 2021.

[26] Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph Turian. Experience grounds language. In *EMNLP*, 2020.

[27] Yonatan Bisk, Daniel Marcu, and William Wong. Towards a dataset for human computer communication via grounded language acquisition. In *AAAI Workshop on Symbiotic Cognitive Systems*, 2016.

[28] Yonatan Bisk, Deniz Yuret, and Daniel Marcu. Natural language communication with robots. In *NAACL*, 2016.

[29] Valts Blukis, Ross A. Knepper, and Yoav Artzi. Few-shot object grounding for mapping natural language instructions to robot control. In *CoRL*, 2020.

[30] Valts Blukis, Dipendra Misra, Ross A. Knepper, and Yoav Artzi. Mapping navigation instructions to continuous control actions with position-visitation prediction. In *CoRL*, 2018.

[31] Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *CoRL*, 2022.

[32] Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer, 2013.

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

[34] Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. Audiolm: a language modeling approach to audio generation. *arXiv:2209.03143*, 2022.

[35] Rodney A Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, 1991.

[36] Rodney A Brooks. New approaches to robotics. *Science*, 1991.

[37] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020.

[38] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[39] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[40] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from RGB-D data in indoor environments. *3DV*, 2017.

[41] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable spatial planning using transformers. In *ICML*, 2021.

[42] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2017.

[43] David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, 2011.

[44] Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *CVPR*, 2019.

[45] 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 NeurIPS*, 2021.

[46] Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. 2020.

[47] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[48] Ting Chen, Saurabh Saxena, Lala Li, David J Fleet, and Geoffrey Hinton. Pix2seq: A language modeling framework for object detection. *arXiv:2109.10852*, 2021.

[49] Yen Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *ECCV*, 2020.

[50] Yiye Chen, Ruinian Xu, Yunzhi Lin, and Patricio A. Vela. A Joint Network for Grasp Detection Conditioned on Natural Language Commands. *arXiv:2104.00492*, April 2021.

[51] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *ICLR*, 2019.

[52] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP*, 2014.

[53] Andy Clark. *Surfing uncertainty: Prediction, Action, and the Embodied Mind*. Oxford University Press, 2015.

[54] Henry M Clever, Ankur Handa, Hammad Mazhar, Kevin Parker, Omer Shapira, Qian Wan, Yashraj Narang, Iretiayo Akinola, Maya Cakmak, and Dieter Fox. Assistive teleop: Leveraging transformers to collect robotic task demonstrations. *arXiv:2112.05129*, 2021.

[55] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *ICML*, 2016.

[56] Rodolfo Corona, Shizhan Zhu, Dan Klein, and Trevor Darrell. Voxel-informed language grounding. In *ACL*, 2022.

[57] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018.

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

[59] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *ECCV*, 2018.

[60] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018.

[61] Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural Modular Control for Embodied Question Answering. In *CoRL*, 2018.

[62] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José M.F. Moura, Devi Parikh, and Dhruv Batra. Visual Dialog. In *CVPR*, 2017.

[63] Abhishek Das, Satwik Kottur, José M.F. Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In *ICCV*, 2017.

[64] Sudeep Dasari and Abhinav Gupta. Transformers for one-shot visual imitation. *arXiv:2011.05970*, 2020.

[65] Harm De Vries, Dzmitry Bahdanau, and Christopher Manning. Towards ecologically valid research on language user interfaces. *arXiv:2007.14435*, 2020.

[66] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[67] Xinke Deng, Yu Xiang, Arsalan Mousavian, Clemens Eppner, Timothy Bretl, and Dieter Fox. Self-supervised 6d object pose estimation for robot manipulation. In *ICRA*, 2020.

[68] AM Derrington and P Lennie. Spatial and temporal contrast sensitivities of neurones in lateral geniculate nucleus of macaque. *The Journal of Physiology*, 357(1):219–240, 1984.

[69] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2018.

[70] David Ding, Felix Hill, Adam Santoro, and Matt Botvinick. Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures. *arXiv:2012.08508*, 2020.

[71] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.

[72] Guy Dove. Thinking in words: language as an embodied medium of thought. *Topics in Cognitive Science*, 6(3):371–389, 2014.

[73] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *ICRA*, 2022.

[74] 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. *arXiv:2303.03378*, 2023.

[75] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634, 2018.

[76] Evelina Fedorenko and Rosemary Varley. Language and thought are not the same thing: evidence from neuroimaging and neurological patients. *Annals of the New York Academy of Sciences*, 1369(1):132–153, 2016.

[77] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *CVPR*, 2019.

[78] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 2016.

[79] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *ICRA*, 2017.

[80] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *CoRL*, 2017.

[81] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *CoRL*, 2022.

[82] Peter Florence, Lucas Manuelli, and Russ Tedrake. Self-supervised correspondence in visuomotor policy learning. *RA-L*, 2019.

[83] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *CoRL*, 2018.

[84] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. In *NeurIPS*, 2018.

[85] Alberto Garcia-Garcia, Pablo Martinez-Gonzalez, Sergiu Oprea, John Alejandro Castro-Vargas, Sergio Orts-Escolano, Jose Garcia-Rodriguez, and Alvaro Jover-Alvarez. The robotrix: An extremely photorealistic and very-large-scale indoor dataset of sequences with robot trajectories and interactions. In *IROS*, 2018.

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

[87] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. Bottom-up abstractive summarization. In *EMNLP*, 2018.

[88] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl the planning domain definition language. 1998.

[89] James J Gibson. *The Ecological Approach to Visual Perception*. Psychology Press, 1979.

[90] Gabriel Goh, Cammarata Nick, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 2021. https://distill.pub/2021/multimodal-neurons.

[91] Daniel Gordon, Dieter Fox, and Ali Farhadi. What should i do now? marrying reinforcement learning and symbolic planning. *arXiv:1901.01492*, 2019.

[92] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *CVPR*, 2018.

[93] Anirudh Goyal, Aniket Rajiv Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Curtis Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. In *ICLR*, 2021.

[94] Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using natural language for reward shaping in reinforcement learning. In *IJCAI*, 2019.

[95] Jonathan Gray, Kavya Srinet, Yacine Jernite, Haonan Yu, Zhuoyuan Chen, Demi Guo, Siddharth Goyal, C. Lawrence Zitnick, and Arthur Szlam. Craftassist: A framework for dialogue-enabled interactive agents, 2019.

[96] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. In *ACL*, 2016.

[97] Agrim Gupta, Linxi Fan, Surya Ganguli, and Li Fei-Fei. Metamorph: Learning universal controllers with transformers. *arXiv:2203.11931*, 2022.

[98] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*. 2018.

[99] Yunhai Han, Rahul Batra, Nathan Boyd, Tuo Zhao, Yu She, Seth Hutchinson, and Ye Zhao. Learning generalizable vision-tactile robotic grasping strategy for deformable objects via transformer. *arXiv:2112.06374*, 2021.

[100] Stevan Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[101] Jun Hatori, Yuta Kikuchi, Sosuke Kobayashi, Kuniyuki Takahashi, Yuta Tsuboi, Yuya Unno, Wilson Ko, and Jethro Tan. Interactively picking real-world objects with unconstrained spoken language instructions. In *ICRA*, 2018.

[102] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv:1507.06527*, 2015.

[103] Matthew J Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *AAAI*, 2020.

[104] Chenhang He, Ruihuang Li, Shuai Li, and Lei Zhang. Voxel set transformer: A set-to-set approach to 3d object detection from point clouds. In *CVPR*, pages 8417–8427, 2022.

[105] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, pages 9729–9738, 2020.

[106] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *CVPR*, 2017.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[108] Malte Helmert. The Fast Downward planning system. *JAIR*, 2006.

[109] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv:2204.03458*, 2022.

[110] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv:2203.15556*, 2022.

[111] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *JAIR*, 2001.

[112] Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions. In *NeurIPS*. 2019.

[113] Ronghang Hu, Marcus Rohrbach, Jacob Andreas, Trevor Darrell, and Kate Saenko. Modeling relationships in referential expressions with compositional modular networks. In *CVPR*, 2017.

[114] Ronghang Hu, Huazhe Xu, Marcus Rohrbach, Jiashi Feng, Kate Saenko, and Trevor Darrell. Natural Language Object Retrieval. *arXiv:1511.04164*, 2016.

[115] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv:2201.07207*, 2022.

[116] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of Neurophysiology*, 28(2):229–289, 1965.

[117] Yuki Inoue and Hiroki Ohashi. Prompter: Utilizing large language model prompting for a data efficient embodied instruction following. *arXiv:2211.03267*, 2022.

[118] Jana M Iverson. Developing language in a developing body: The relationship between motor development and language development. *Journal of Child Language*, 37(2):229–261, 2010.

[119] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv:2107.14795*, 2021.

[120] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *ICML*, 2021.

[121] Unnat Jain, Alexander Schwing, and Svetlana Lazebnik. Two can play this game: Visual dialog with discriminative question generation and answering. In *CVPR*, 2018.

[122] Stephen James and Pieter Abbeel. Bingham policy parameterization for 3d rotations in reinforcement learning. *arXiv:2202.03957*, 2022.

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

[124] Stephen James and Pieter Abbeel. Coarse-to-fine q-attention with tree expansion. *arXiv:2204.12471*, 2022.

[125] Stephen James and Andrew J. Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *RA-L*, 2022.

[126] Stephen James, Marc Freese, and Andrew J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv:1906.11176*, 2019.

[127] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *RA-L*, 2020.

[128] Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *CVPR*, 2022.

[129] 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 *CoRL*, 2021.

[130] Eric Jang, Sudheendra Vijayanarasimhan, Peter Pastor, Julian Ibarz, and Sergey Levine. End-to-end learning of semantic grasping. In *CoRL*, 2017.

[131] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *NeurIPS*, 2021.

[132] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *NeurIPS*, 2019.

[133] Edward Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In *ICRA*, 2021.

[134] Jacob J Johnson, Linjun Li, Ahmed H Qureshi, and Michael C Yip. Motion planning transformers: One model to plan them all. *arXiv:2106.02791*, 2021.

[135] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

[136] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *CVPR*, 2016.

[137] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

[138] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011.

[139] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *IJRR*, 2013.

[140] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRL*, 2018.

[141] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv:2104.08212*, 2021.

[142] Aishwarya Kamath, Mannat Singh, Yann LeCun, Ishan Misra, Gabriel Synnaeve, and Nicolas Carion. Mdetr–modulated detection for end-to-end multi-modal understanding. *arXiv:2104.12763*, 2021.

[143] Evangelos Kazakos, Arsha Nagrani, Andrew Zisserman, and Dima Damen. Slow-fast auditory streams for audio recognition. In *ICASSP*, 2021.

[144] Liyiming Ke, Xiujun Li, Yonatan Bisk, Ari Holtzman, Zhe Gan, Jingjing Liu, Jianfeng Gao, Yejin Choi, and Siddhartha Srinivasa. Tactical rewind: Self-correction via backtracking in vision-and-language navigation. In *CVPR*, 2019.

[145] Heecheol Kim, Yoshiyuki Ohmura, and Yasuo Kuniyoshi. Transformer-based deep imitation learning for dual-arm robot manipulation. In *IROS*, 2021.

[146] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[147] Jan Koenderink. *Sentience*. De Clootcrans Press, 2019.

[148] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv:1712.05474*, 2017.

[149] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *ICML*, 2018.

[150] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *JAIR*, 2018.

[151] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. In *NeurIPS*, 2012.

[152] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. *NeurIPS*, 2019.

[153] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, pages 3675–3683, 2016.

[154] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NeurIPS*, 2015.

[155] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment, 2020.

[156] Shamit Lal, Mihir Prabhudesai, Ishita Mediratta, Adam W Harley, and Katerina Fragkiadaki. Coconets: Continuous contrastive 3d scene representations. In *CVPR*, 2021.

[157] Nathan Lambert, Louis Castricato, Leandro von Werra, and Alex Havrilla. Illustrating reinforcement learning from human feedback (rlhf). *Hugging Face Blog*, 2022. https://huggingface.co/blog/rlhf.

[158] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[159] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[160] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv:2205.15241*, 2022.

[161] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(1):1334–1373, 2016.

[162] Klaus Libertus and Dominic A Violi. Sit to talk: Relation between motor skills and language development in infancy. *Frontiers in Psychology*, 7:475, 2016.

[163] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[164] Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-lambda: Disentangling dynamic task relationships. *TMLR*, 2022.

[165] Weiyu Liu, Chris Paxton, Tucker Hermans, and Dieter Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *ICRA*, 2022.

[166] Xingyu Liu, Rico Jonschkowski, Anelia Angelova, and Kurt Konolige. Keypose: Multi-view 3d labeling and keypoint estimation for transparent objects. In *CVPR*, 2020.

[167] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.11692*, 2019.

[168] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021.

[169] Margaret Livingstone and David Hubel. Segregation of form, color, movement, and depth: anatomy, physiology, and perception. *Science*, 240(4853):740–749, 1988.

[170] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *NeurIPS*, 2020.

[171] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *NeurIPS*, 2019.

[172] Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 12-in-1: Multi-task vision and language representation learning. In *CVPR*, June 2020.

[173] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *IJCAI*, 2019.

[174] Corey Lynch and Pierre Sermanet. Grounding language in play. *arXiv:2005.07648*, 2020.

[175] Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming Xiong. Self-monitoring navigation agent via auxiliary progress estimation. In *ICLR*, 2019.

[176] Chih-Yao Ma, Zuxuan Wu, Ghassan AlRegib, Caiming Xiong, and Zsolt Kira. The regretful agent: Heuristic-aided navigation through progress estimation. In *CVPR*, 2019.

[177] James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda Muresan, Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding english commands to reward functions. In *RSS*, 2015.

[178] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI*, 2006.

[179] Jonathan Malmaud, Earl Wagner, Nancy Chang, and Kevin Murphy. Cooking with semantics. In *ACL Workshop on Semantic Parsing*, 2014.

[180] Zhao Mandi, Pieter Abbeel, and Stephen James. On the effectiveness of fine-tuning versus meta-reinforcement learning. *arXiv:2206.03271*, 2022.

[181] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. In *ISRR*, 2019.

[182] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. In *ICCV*, 2021.

[183] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv:1904.12584*, 2019.

[184] Alana Marzoev, Samuel Madden, M Frans Kaashoek, Michael Cafarella, and Jacob Andreas. Unnatural language processing: Bridging the gap between synthetic and natural language data. *arXiv:2004.13645*, 2020.

[185] Daniela Massiceti, N. Siddharth, Puneet Kumar Dokania, and Philip H. S. Torr. Flipdial: A generative model for two-way visual dialogue. In *CVPR*, 2018.

[186] Cynthia Matuszek, Liefeng Bo, Luke Zettlemoyer, and Dieter Fox. Learning from unscripted deictic gesture and language for human-robot interactions. In *AAAI*, 2014.

[187] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[188] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *RA-L*, 2022.

[189] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *arXiv:2112.03227*, 2021.

[190] So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods, 2021.

[191] Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In *EMNLP*, 2018.

[192] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017.

[193] Dipendra Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. In *RSS*, 2014.

[194] Dipendra Misra, Kejia Tao, Percy Liang, and Ashutosh Saxena. Environment-driven lexicon induction for high-level instructions. In *ACL*, 2015.

[195] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *IJRR*, 35(1-3):281–300, 2016.

[196] H Moravec. Robot spatial perceptionby stereoscopic vision and 3d evidence grids. *Perception*, 1996.

[197] Hans Moravec. *Mind Children: The Future of Robot and Human intelligence*. Harvard University Press, 1988.

[198] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *ICCV*, 2019.

[199] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ToG*, 2022.

[200] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. 6-dof grasping for target-driven object manipulation in clutter. In *ICRA*, 2020.

[201] Lakshmi Nair, Jonathan C. Balloch, and Sonia Chernova. Tool macgyvering: Tool construction using geometric reasoning. In *ICRA*, 2019.

[202] Suraj Nair, Eric Mitchell, Kevin Chen, Silvio Savarese, Chelsea Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *CoRL*, 2022.

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

[204] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning. *JAIR*, 63(1):849–874, 2018.

[205] Khanh Nguyen and Hal Daumé III. Help, Anna! Visual Navigation with Natural Multimodal Assistance via Retrospective Curiosity-Encouraging Imitation Learning. In *EMNLP*, 2019.

[206] Khanh Nguyen, Debadeepta Dey, Chris Brockett, and Bill Dolan. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. In *CVPR*, 2019.

[207] Thao Nguyen, Nakul Gopalan, Roma Patel, Matthew Corsaro, Ellie Pavlick, and Stefanie Tellex. Robot object retrieval with contextual natural language queries. In *RSS*, Corvalis, Oregon, USA, July 2020.

[208] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ToG*, 2013.

[209] Daniel Nyga, Subhro Roy, Rohan Paul, Daehyung Park, Mihai Pomarlan, Michael Beetz, and Nicholas Roy. Grounding robot plans from natural language instructions with incomplete world knowledge. In *CoRL*, 2018.

[210] OpenAI. Gpt-4 technical report, 2023.

[211] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 2022.

[212] Andrew Parker. In the blink of an eye: How vision sparked the big bang of evolution. 2003.

[213] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*.

[214] Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic transformer for vision-and-language navigation. In *ICCV*, 2021.

[215] Ramakanth Pasunuru and Mohit Bansal. Game-based video-context dialogue. In *EMNLP*, 2018.

[216] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv:1912.01703*, 2019.

[217] Rohan Paul, Jacob Arkin, Derya Aksaray, Nicholas Roy, and Thomas M. Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot platforms. *IJRR*, 2018.

[218] Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. *RSS*, 2016.

[219] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Fox. Prospection: Interpretable plans from language by predicting the future. In *ICRA*, 2019.

[220] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

[221] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv:2209.14988*, 2022.

[222] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv:1608.05859*, 2016.

[223] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *CVPR*, 2018.

[224] 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. *arXiv:2103.00020*, February 2021.

[225] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv:2204.06125*, 2022.

[226] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *ICCV*, 2021.

[227] D Raj Reddy et al. Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science. Carnegie Mellon University, Pittsburgh, PA*, 17, 1977.

[228] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv:2205.06175*, 2022.

[229] Michaela Regneri, Marcus Rohrbach, Dominikus Wetzel, Stefan Thater, Bernt Schiele, and Manfred Pinkal. Grounding action descriptions in videos. *TACL*, 2013.

[230] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *IROS*, 2013.

[231] Marcus Rohrbach, Sikandar Amin, Mykhaylo Andriluka, and Bernt Schiele. A database for fine grained activity detection of cooking activities. In *CVPR*, 2012.

[232] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[233] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[234] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[235] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, 1989.

[236] James Owen Ryan, Michael Mateas, and Noah Wardrip-Fruin. Characters who speak their minds: Dialogue generation in talk of the town. In *AAAI-AIIDE*, 2016.

[237] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv:1910.01108*, 2019.

[238] Shibani Santurkar, Yann Dubois, Rohan Taori, Percy Liang, and Tatsunori Hashimoto. Is a caption worth a thousand images? a controlled study for representation learning. *arXiv:2207.07635*, 2022.

[239] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022.

[240] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *ICCV*, 2019.

[241] Erez Schwartz, Guy Tennenholtz, Chen Tessler, and Shie Mannor. Language is power: Representing states using natural language in reinforcement learning, 2019.

[242] I. Schwartz, S. Yu, T. Hazan, and A. G. Schwing. Factor graph attention. In *CVPR*, 2019.

[243] Daniel Seita, Pete Florence, Jonathan Tompson, Erwin Coumans, Vikas Sindhwani, Ken Goldberg, and Andy Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks. In *ICRA*, 2021.

[244] Ozan Sener, Amir R. Zamir, Silvio Savarese, and Ashutosh Saxena. Unsupervised semantic parsing of video collections. In *ICCV*, 2015.

[245] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. *arXiv:2204.05186*, 2022.

[246] Shikhar Sharma, Layla El Asri, Hannes Schulz, and Jeremie Zumer. Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *arXiv:1706.09799*, 2017.

[247] Kenneth Shaw, Shikhar Bahl, and Deepak Pathak. Videodex: Learning dexterity from internet videos. In *CoRL*, 2022.

[248] Sheng Shen, Liunian Harold Li, Hao Tan, Mohit Bansal, Anna Rohrbach, Kai-Wei Chang, Zhewei Yao, and Kurt Keutzer. How much can clip benefit vision-and-language tasks? *arXiv:2107.06383*, 2021.

[249] Mohit Shridhar and David Hsu. Interactive visual grounding of referring expressions for human-robot interaction. In *RSS*, 2018.

[250] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *In CoRL*, 2021.

[251] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *In CoRL*, 2022.

[252] 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 *CVPR*, 2020.

[253] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *ICLR*, 2021.

[254] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[255] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. *arXiv:2112.05124*, 2021.

[256] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *arXiv:1406.2199*, 2014.

[257] Kunal Pratap Singh, Suvaansh Bhambri, Byeonghwi Kim, Roozbeh Mottaghi, and Jonghyun Choi. Factorizing perception and policy for interactive instruction following. In *ICCV*, 2021.

[258] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019.

[259] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In Marina Meila and Tong Zhang, editors, *ICML*, 2021.

[260] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser. Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *RA-L*, 5(3):4978–4985, 2020.

[261] Elias Stengel-Eskin, Andrew Hundt, Zhuohong He, Aditya Murali, Nakul Gopalan, Matthew Gombolay, and Gregory Hager. Guiding multi-step rearrangement tasks with natural language instructions. In *CoRL*, 2022.

[262] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *ICCV*, 2019.

[263] Priya Sundaresan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E Gonzalez, and Ken Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In *ICRA*, 2020.

[264] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In *EMNLP*, 2019.

[265] Hao Tan, Licheng Yu, and Mohit Bansal. Learning to navigate unseen environments: Back translation with environmental dropout. In *NAACL*, 2019.

[266] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[267] Kaihua Tang, Yulei Niu, Jianqiang Huang, Jiaxin Shi, and Hanwang Zhang. Unbiased scene graph generation from biased training. In *CVPR*, 2020.

[268] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, 2020.

[269] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55, 2020.

[270] Stefanie Tellex, Ross Knepper, Adrian Li, Daniela Rus, and Nicholas Roy. Asking for help using inverse semantics. 2014.

[271] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[272] Moritz Tenorth, Jan Bandouch, and Michael Beetz. The TUM Kitchen Data Set of Everyday Manipulation Activities for Motion Tracking and Action Recognition. In *ICCV Workshop on Tracking Humans for the Evaluation of their Motion in Image Sequences*, 2009.

[273] Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *ICRA*, 2010.

[274] Jesse Thomason, Daniel Gordon, and Yonatan Bisk. Shifting the baseline: Single modality performance on visual navigation & qa. In *NAACL*, 2019.

[275] Jesse Thomason, Michael Murray, Maya Cakmak, and Luke Zettlemoyer. Vision-and-dialog navigation. In *CoRL*, 2019.

[276] Jesse Thomason, Jivko Sinapov, Maxwell Svetlik, Peter Stone, and Raymond J Mooney. Learning multi-modal grounded linguistic semantics by playing “i spy". In *IJCAI*, 2016.

[277] Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. Learning to interpret natural language commands through human-robot dialog. In *IJCAI*, 2015.

[278] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

[279] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators. *arXiv:2011.06464*, 2020.

[280] Tomer D Ullman, Elizabeth Spelke, Peter Battaglia, and Joshua B Tenenbaum. Mind games: Game engines as an architecture for intuitive physics. *Trends in Cognitive Sciences*, 2017.

[281] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*. 2017.

[282] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, 2, 2019.

[283] Chen Wang, Linxi Fan, Jiankai Sun, Ruohan Zhang, Li Fei-Fei, Danfei Xu, Yuke Zhu, and Anima Anandkumar. Mimicplay: Long-horizon imitation learning by watching human play. *arXiv:2302.12422*, 2023.

[284] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *CVPR*, 2019.

[285] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *ECCV*, 2018.

[286] Erik Wijmans, Samyak Datta, Oleksandr Maksymets, Abhishek Das, Georgia Gkioxari, Stefan Lee, Irfan Essa, Devi Parikh, and Dhruv Batra. Embodied question answering in photorealistic environments with point cloud perception. In *CVPR*, 2019.

[287] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MIT Cambridge Project MAC, 1971.

[288] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *NeurIPS*, 2017.

[289] Qi Wu, Peng Wang, Chunhua Shen, Ian D. Reid, and Anton van den Hengel. Are you talking to me? reasoned visual dialog generation through adversarial learning. In *CVPR*, 2017.

[290] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to Manipulate Deformable Objects without Demonstrations. In *RSS*, 2020.

[291] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *CVPR*, 2018.

[292] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *RSS*, 2018.

[293] Fanyi Xiao, Yong Jae Lee, Kristen Grauman, Jitendra Malik, and Christoph Feichtenhofer. Audiovisual slowfast networks for video recognition. *arXiv:2001.08740*, 2020.

[294] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *CoRL*, 2020.

[295] Haoyu Xiong, Quanzhou Li, Yun-Chun Chen, Homanga Bharadhwaj, Samarth Sinha, and Animesh Garg. Learning by watching: Physical imitation of manipulation skills from human videos. In *IROS*, 2021.

[296] Zhenjia Xu, He Zhanpeng, and Shuran Song. Umpnet: Universal manipulation policy network for articulated objects. *RA-L*, 2022.

[297] Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. *arXiv:2107.03996*, 2021.

[298] Yiqun Yao, Jiaming Xu, and Bo Xu. The world in my mind: Visual dialog with adversarial multi-modal feature encoding. In *NAACL*, 2019.

[299] Lin Yen-Chen, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. Learning to see before learning to act: Visual pre-training for manipulation. In *ICRA*, 2020.

[300] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv:1904.00962*, 2019.

[301] Adams Wei Yu, David Dohan, Quoc Le, Thang Luong, Rui Zhao, and Kai Chen. Fast and accurate reading comprehension by combining self-attention and convolution. In *ICLR*, 2018.

[302] Dong Yu and Li Deng. *Automatic Speech Recognition*, volume 1. Springer, 2016.

[303] Fei Yu, Jiji Tang, Weichong Yin, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. Ernie-vil: Knowledge enhanced vision-language representations through scene graph. *arXiv:2006.16934*, 2020.

[304] Haonan Yu and Jeffrey Mark Siskind. Grounded language learning from video described with sentences. In *ACL*, 2013.

[305] Licheng Yu, Xinlei Chen, Georgia Gkioxari, Mohit Bansal, Tamara L. Berg, and Dhruv Batra. Multi-target embodied question answering. In *CVPR*, 2019.

[306] Licheng Yu, Zhe Lin, Xiaohui Shen, Jimei Yang, Xin Lu, Mohit Bansal, and Tamara L Berg. Mattnet: Modular attention network for referring expression comprehension. In *CVPR*, 2018.

[307] 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 *CoRL*, 2020.

[308] Wentao Yuan, Chris Paxton, Karthik Desingh, and Dieter Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *In CoRL*, 2021.

[309] Xingdi Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. Counting to explore and generalize in text-based games. *arXiv:1806.11525*, 2018.

[310] Kevin Zakka, Andy Zeng, Johnny Lee, and Shuran Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. In *ICRA*, 2020.

[311] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *CoRL*, 2020.

[312] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *T-RO*, 2020.

[313] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *ICRA*, 2018.

[314] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *IJRR*, 2019.

[315] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv:2204.00598*, 2022.

[316] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In *ICRA*, 2017.

[317] Yichi Zhang and Joyce Chai. Hierarchical task learning from language instructions with unified transformers and self-monitoring. In *ACL-IJCNLP*, 2021.

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

[319] Kaiyu Zheng, Rohan Chitnis, Yoonchang Sung, George Konidaris, and Stefanie Tellex. Towards optimal correlational object search. In *ICRA*, 2022.

[320] Zilong Zheng, Wenguan Wang, Siyuan Qi, and Song-Chun Zhu. Reasoning visual dialogs with structural and partial observations. In *CVPR*, 2019.

[321] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. RTFM: Generalising to novel environment dynamics via reading. In *ICLR*, 2020.

[322] Menglong Zhu, Konstantinos G Derpanis, Yinfei Yang, Samarth Brahmbhatt, Mabel Zhang, Cody Phillips, Matthieu Lecce, and Kostas Daniilidis. Single image 3d object detection and pose estimation for grasping. In *ICRA*, 2014.

[323] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual semantic planning using deep successor representations. In *ICCV*, 2017.

[324] Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. Cross-task weakly supervised learning from instructional videos. In *CVPR*, 2019.

Appendix A

ALFRED

A.1 Dataset Details

We give additional information about the generation of expert demonstrations in AI2-THOR, language directives, the annotation interface used to collect directives, and samples of annotations with their associated demonstrations.

A.1.1 Expert Demonstrations

When sampling task parameters, we employ an active strategy to maximize data heterogeneity. Figure A.1 shows the distribution of high-level task across train, validation seen, and validation unseen folds. Figure A.1 shows the distribution of subgoals across task types. And Figure A.5 and Figure A.6 give the distributions of pickup objects and receptacles across the dataset. Each task parameter sample is defined by (t, s, o, r, m), where

- $t =$ the task type;
- $s =$ the scene in AI2-THOR;
- $o =$ the object class to be picked up;
- $r =$ the final destination for o or $\emptyset$ for **Examine**;
- $m =$ the secondary object class for **Stack & Place** tasks ($\emptyset$ for other task types).

To construct the next tuple, we first find the largest source of imbalance in the current set of tuples. For example if $o = apple$ is more common than $o = plunger$, $o = plunger$ will be ranked higher than $o = apple$. We additionally account for the prior distribution of each entity (e.g.,, if *cup* is already represented in the data often as both o and m, it becomes disfavored by the sampling algorithm for all slots). We do this greedily across all slots until the tuple is complete. Given any partial piece of information about the task, the distributions of the remaining task parameters remain heterogeneous under this sampling, weakening baseline priors such as ignoring the language input and always executing a common task in the environment.

Once a task parameter sample is complete, the chosen scene is instantiated, objects and agent start position are randomized, and the relevant room data is encoded into PDDL rules for an expert demonstration. If the PDDL planner cannot generate an expert demonstration given the room configuration, or if the agent fails an action during execution, for example by running into walls or opening doors onto itself due to physical constraints, the episode

FIGURE A.1: **Task and Subgoal Distributions.** *(Left)* Task distribution across train, validation seen and unseen dataset splits. *(Right)* Subgoal distribution across 7 task types.

is abandoned. We gather three distinct expert demonstrations per task parameter sample. These demonstrations are further vetted by rolling them forward using our wrapper to the AI2-THOR API to ensure that a "perfect" model can reproduce the demonstration. The full sampling generation and verification code will be published along with the dataset.

A.1.2 Example Language Directives

We chose to gather three directives per demonstration empirically. For a subset of over 700 demonstrations, we gathered up to 6 language directives from different annotators. We find that after three annotations, fewer than 10 unique tokens on average are introduced by additional annotators (Figure A.2).

A.1.3 Annotation Interface

Figure A.3 shows the Mechanical Turk interface used to gather language annotations. Workers were presented with a video of the expert demonstration with timeline segments indicating sub-goals. The workers annotated each segment while scrubbing through the video, and wrote a short summary description for the entire sequence. We payed workers \$0.7 per annotation. During vetting, annotators were paid \$0.35 per HIT (Human Interaction Task) to compare 5 sets of three directives each. These wages were set based on local minimum-wage rates and average completion time.

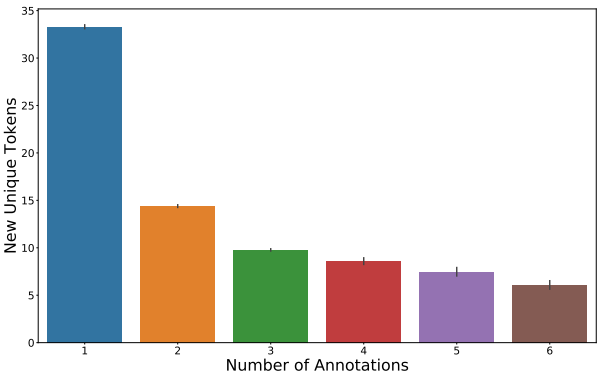


FIGURE A.2: The number of unique tokens introduced per annotation of language directives.

A.1.4 Vocabulary Distributions

Figure A.7 shows vocabulary statistics of the language in ALFRED.

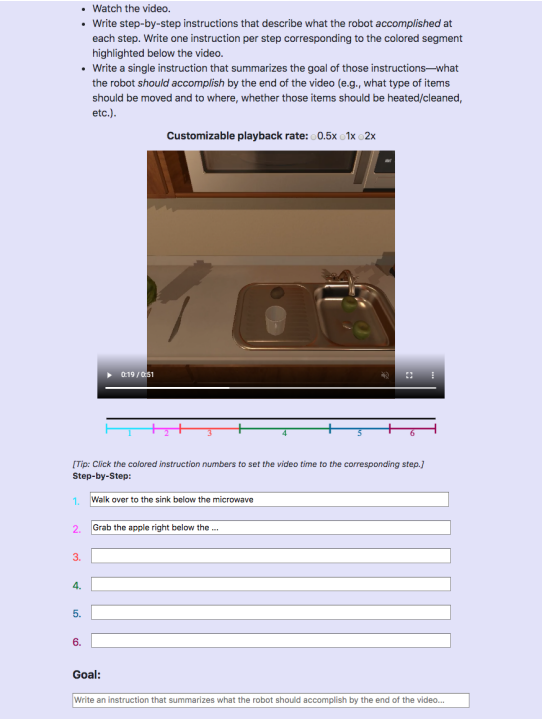


FIGURE A.3: **Mechanical Turk Interface**. Annotation data collection setup.

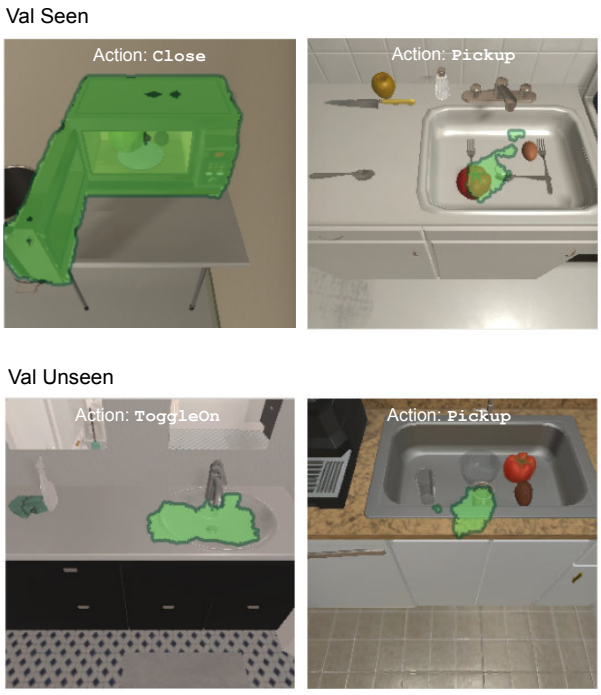


FIGURE A.4: **Predicted Interaction Masks.** Masks generated by the SEQ2SEQ+PM model are displayed in green.

A.1.5 Dataset Examples

Figure A.8 shows 7 expert trajectories (one per task type) and their accompanied annotations.

A.2 Implementation Details

We describe implementation and training details of our baseline Sequence-to-Sequence models.

Preprocessing We tokenize the language directives and convert all tokens to lower-case. During dataset generation, we save images from AI2-THOR 300×300 pixels, and later resize them to 224×224 during training. The generation pipeline saves initialization information for objects and the agent, so all demonstration can be perfectly replayed in the simulator. Researchers can use this replay feature to augment the dataset by saving high-res images, depth maps, or object-segmentation masks.

Network Architecture We use a pretrained ResNet-18 [107] to extract $512 \times 7 \times 7$ features from the `conv5` layer. These features are fed into a two-layer CNN with 1×1 convolutions to reduce the channel dimension from 512 to 64. The $64 \times 7 \times 7$ output is flattened, and a fully-connected layer produces a 2500-dimensional visual feature v_t.

Task Ablations - **Validation**						
	NO LANGUAGE		SEQ2SEQ		SEQ2SEQ+PM	
Task Type	*Seen*	*Unseen*	*Seen*	*Unseen*	*Seen*	*Unseen*
Pick & Place	0.0	0.0	6.3	**1.0**	**7.0**	0.0
Stack & Place	0.0	0.0	0.0	0.0	**0.9**	0.0
Pick Two	0.0	0.0	**1.6**	0.0	0.8	0.0
Clean & Place	0.0	0.0	0.0	0.0	1.8	0.0
Heat & Place	0.0	0.0	**1.9**	0.0	**1.9**	0.0
Cool & Place	0.0	0.0	2.4	0.0	**4.0**	0.0
Examine	0.0	0.0	4.3	0.0	**9.6**	0.0

TABLE A.1: **Success Percentages Across 7 Task Types.** The highest values are shown in **blue**.

The language encoder is a bi-directional LSTM with a hidden-dimension of 100. We do not use pretrained language models to initialize the LSTM, and the encodings are learned from scratch in an end-to-end manner. We also use a self-attention mechanism to attend over the encodings to initialize the hidden-state of the decoder LSTM.

The action decoder is an LSTM with a hidden-dimension of 512. The actor is a fully-connected layer that outputs logits for 13 actions. The mask decoder is a three-layer deconvolution network, which takes in the concatenated vector u_t and transforms it into $64 \times 7 \times 7$ features with a fully-connected layer. These features are subsequently up-scaled into a $1 \times 300 \times 300$ binary mask through three layers of deconvolutions and up-sampling with bi-linear interpolation.

Training The models were implemented with PyTorch and trained with the Adam optimizer [146] at a learning rate of 1e-4. We use dropout of 0.3 on the visual features and the decoder hidden state, tuned on the validation data. Both the action and mask losses are weighted equally, while the auxiliary losses are scaled with a factor of 0.1. For evaluation, we choose models with the lowest loss on the validation *seen* set. It should be noted that, due to the nature of the tasks, low validation loss might not directly lead to better evaluation performance since the agent does not have to exactly imitate the expert to complete the task.

Notes on Random Agent Unlike discretized navigation where taking random actions might allow the agent to stumble upon the goal, ALFRED tasks are much harder to achieve by chance. The action space branching factor of Room-to-Room navigation [13], for example, is $4^6 \approx 4000$ (6 average steps and 4 navigation actions). By contrast, the ALFRED branching factor is $12^{50} \approx 10^{53}$ (50 average steps for 12 actions). Beyond action type prediction, the ALFRED space resulting from dynamic environments and the need to produce pixel-wise masks for interactive actions explodes further.

A.2.1 Predicted Masks

Figure A.4 shows a few examples of masks generated by the SEQ2SEQ+PM model in seen and unseen validation scenes. The *Microwave* mask accurately captures the contours of the object since the model is familiar with receptacles in seen environments. In contrast, the *Sink* mask in the unseen scene poorly fits the unfamiliar object topology.

A.3 Additional Results

A.3.1 Performance by Task Type

In Table A.1, we present success rates across the 7 task types. Even the best performing model, SEQ2SEQ+PM, mostly succeeds in solving some short-horizon tasks like **Pick & Place** and **Examine**. Long horizon tasks like **Stack & Place** and **Pick Two & Place** have near zero success rates across all models.

FIGURE A.5: Pickup distributions in the train, validation *seen* and *unseen* folds.

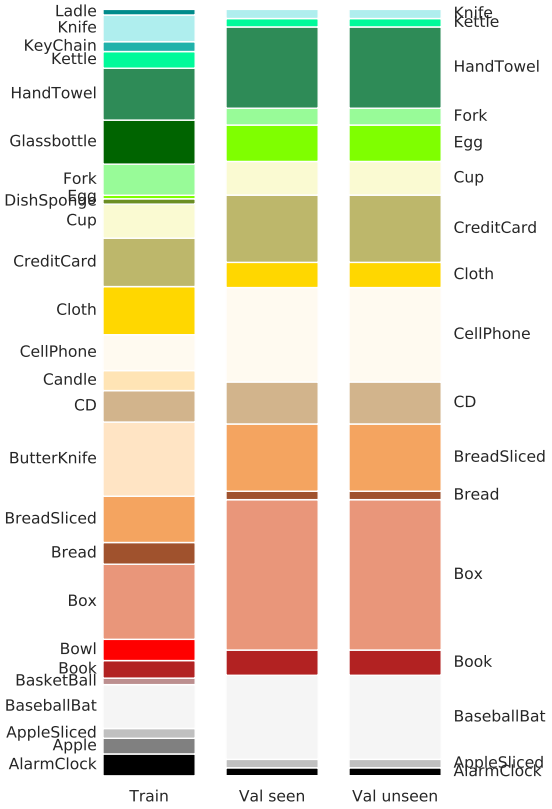


FIGURE A.6: Receptacle distributions in the train, validation *seen* and *unseen* folds.

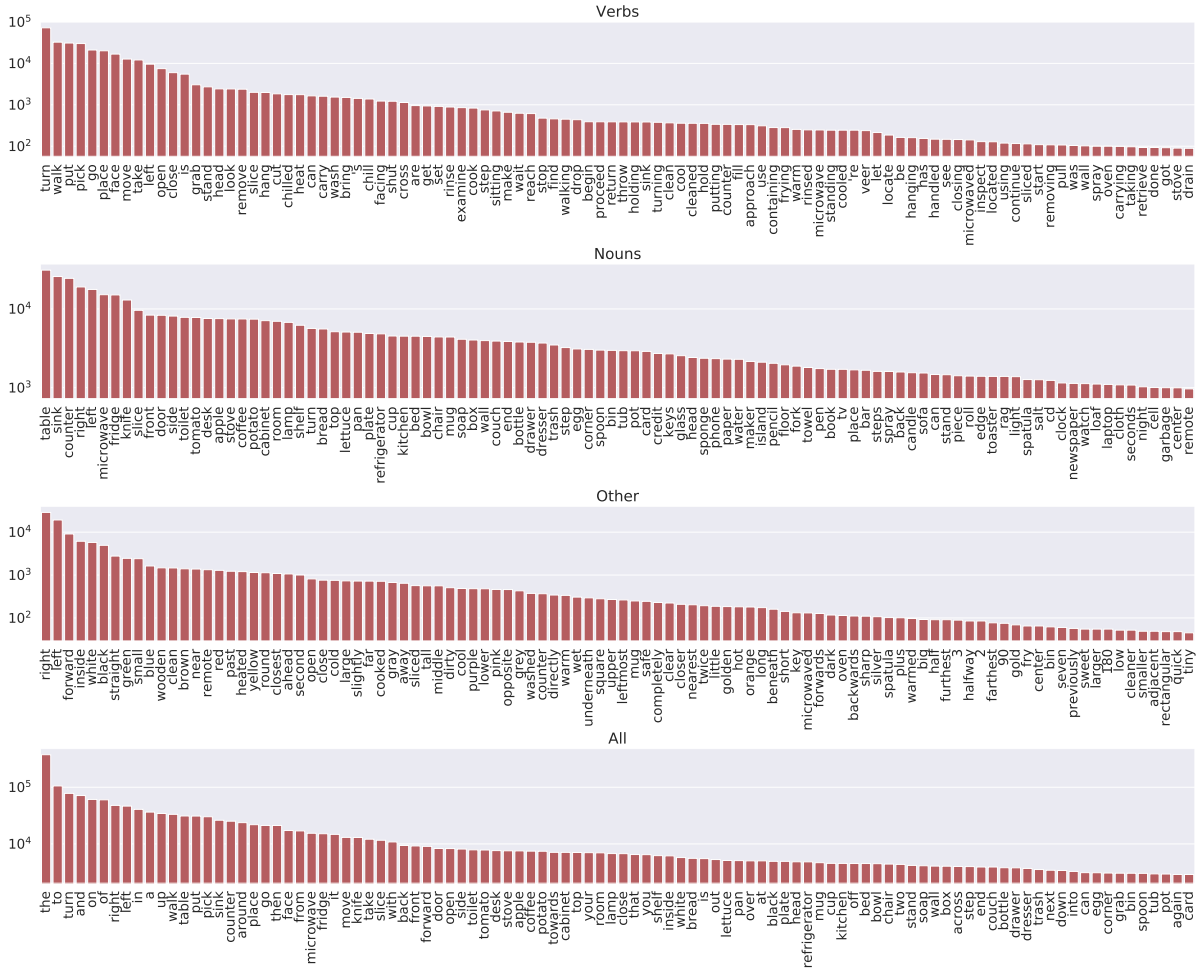
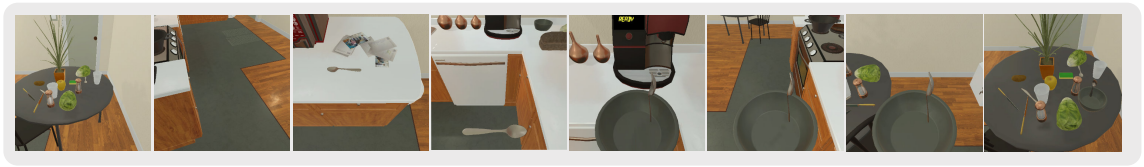


FIGURE A.7: **Vocabulary Distributions.** Frequency distributions of 100 most common verbs, nouns, other words (non-verbs and non-nouns), and all words.

Pick & Place

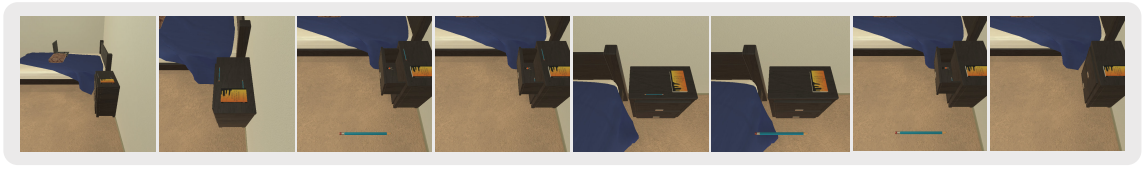
	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Put a watch on a table.**	**Put the watch on the coffee table.**	**Move the watch to the coffee table.**
Instructions	Go to the right and turn around to face the end of the cabinet with the television on it. Pick the watch up from the cabinet. Go to the right and then turn to face the coffee table in front of the couch. Put the watch on the table.	Turn right and go to the TV stand. Pick up the watch from the stand. Turn around and face the coffee table. Put the watch on the coffee table.	Turn right, go straight, turn right, step forward, then turn right to face the table with the TV on it. Pick up the watch on the table, to the right of the remote. Turn right, move forward, turn left, move forward, then turn right to face the coffee table. Put the watch on the front left corner of the coffee table.

Stack & Place



	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Put a bowl with a spoon in it on the table.**	**Move a spoon and bowl to a table.**	**To put a spoon in a bowl plus moving them to the kitchen table.**
Instructions	Turn left, and walk across the room to the microwave. Pick up the spoon. Turn left, and walk to the coffee machine .Put the spoon in the bowl. Pick up the bowl. Turn left, walk to the table, and turn left. Put the bowl on the table.	Move a spoon and bowl to a table. Go to the counter right of the microwave. Pick up the spoon from the counter. Go to the coffeemaker. Place the spoon in a bowl next to the coffeemaker. Pick up the bowl. Go to the black table. Place the bowl down on the table.	Turn left and walk across the room to face a spoon on the right end of the counter. Pick up the spoon on the end of the counter. Turn left and walk across the room to face the coffee maker on the counter. Put the spoon in the bowl to the right of the coffee maker on the counter. Pick up the bowl with a spoon in it on the counter. Turn left and walk across the room and turn left to face the kitchen table. Place the bowl with the spoon in it on the kitchen table.

Pick Two & Place



	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Putting pencils inside of a cabinet.**	**Put two pencils in a drawer.**	**Place the two pencils in the stand.**
Instructions	Walk to the bedside table in front of you. Grab the pencil that's on the table. Move slightly to the left and open the top cabinet of the bedside table. Place the pencil inside the cabinet. Face the front of the bedside table. Grab the pencil off of the bedside table. Open the top cabinet on the table. Place the pencil inside the cabinet and then close it.	Walk straight ahead to the end table. Pick up the blue pencil on the right. Walk around to the front of the end table. Put the pencil in the top drawer of the end table. Look up at the top of the end table. Pick up the pencil on the table. Look down at the drawer. Put the pencil in the drawer on top of the other pencil.	Walk to the stand next to the bed. Grab the pencil from the stand. Open the shelf inside of the stand. Place the pencil in the top shelf of the stand. Close the shelf, walk back to the stand. Grab the other pencil from the stand. Walk back to the stand next to the bed. Place the pencil in the top shelf of the stand.

Clean & Place

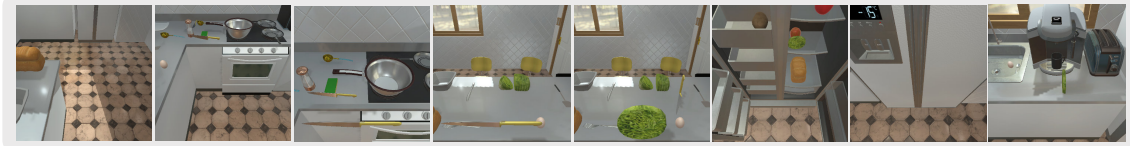
	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Put a clean rag on the top shelf of a barred rack.**	**Wash the pink towel on the shelf, put it back on the shelf.**	**Clean a red cloth.**
Instructions	Turn around, go to the barred rack.Pick up the rag from the bottle shelf of the barred rack. Go to the sink on the left. Put the rag in the sink, turn on then turn off the water. Go to the barred rack to the right of the sink. Put the rag on the top shelf of the barred rack.	Wash the pink towel on the shelf, put it back on the shelf. Turn around and go the shelf. Pick up the pink towel on the shelf. Turn around and put the towel in the sink. Fill the sink with water and wash the towel, take the towel out. Go back to the shelf. Put the towel back on the shelf.	walk on over to the towel drying rack. pick up a dirty red cloth from the towel rack. walk over to the left side of the bathroom sink. turn on the water to rinse the dirty red cloth and pick it back up again. walk back over to the towel drying rack. place the clean cloth on the drying rack.

FIGURE A.8: **Dataset Examples – Part 1.** Annotations for seven expert demonstrations.

Heat & Place

	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Put a cooked potato slice on the counter.**	**Place a slice of cooked potato onto the counter.**	**Put a piece of cooked potato on the counter.**
	Turn right, turn right, walk past the sink, turn left to face round table with tablecloth. Pick up the yellow-handled knife from the table. Cut a slice in the potato on the table. Turn left, turn left, turn right at counter, cross room, turn left at refrigerator to face counter. Put knife down on the table. Turn left, walk past sink, turn left to face round table. Pick the potato slice up from the table. Turn left, make right around corner of counter, turn left to face stove and microwave. Put potato in microwave, cook it, take it out of microwave. Turn right, cross room, turn left at counter with blue plate on it. Put potato on the counter in front of the blue plate.	Turn right, move to the table. Pick up the knife from the table. Slice the potato on the table. Turn left, move to the counter left of the bread. Put the knife on the counter near the soap container. Turn left, move to the table. Pick up a slice of potato from the table. Turn left, move to the counter in front of the stove. Put the potato slice into the microwave, cook it, pick it back up. Turn right, move to the counter left of the bread. Put the cooked potato slice on the counter.	Turn right and cross the room, then turn left and go to face the gray table. Pick up the knife from in between the lettuce and the apple. Use the knife to slice the potato that's on the gray table. Bring the knife with you and go face the kitchen counter with the loaf of bread. Put the knife down in front of the soap dispenser on the counter. Go back over to the gray table. Pick up a slice of the cut potato from the table. Bring the potato with you and go over to the stove, then look up at the microwave. Cook the potato slice in the microwave, then take it out again. Bring the potato slice over to the counter top with the loaf of bread and the knife you used to cut it. Put the potato slice down in front of the blue plate.

Cool & Place



	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Put a slice of cold lettuce on a counter.**	**Put a chilled slice of lettuce on the counter.**	**Slice some lettuce and cool it in the refrigerator so you can put it on the counter top.**
Instructions	Turn left, go forward past the counter, turn left, go forward to the counter to the left of the oven. Take the knife to the left of the large spoon from the counter. Turn around, go forward a step, turn right to the counter. Cut the lettuce on the counter into slices. Turn right, go forward a step, turn left to the counter. Put the knife behind the egg on the counter. Turn left, go forward a step, turn right to the counter. Take a slice of lettuce from the counter. Turn left, go forward, turn right to the fridge. Go to the fridge. Chill the lettuce in the fridge in front of the apple. Take the lettuce from the fridge. Turn left, go forward, turn right to face the coffee maker. Put the lettuce in front of the coffee maker on the counter.	Turn left, head toward the fridge, turn left and go to the stove. Pick up the knife beside the spoon on the counter. Turn around and turn to the right to face the counter with the egg. Cut the lettuce on the counter. Move right and back left to the counter. Put the knife behind the egg on the counter. Move the left and turn back right towards the counter. Pick up a slice of lettuce. Turn around and head to the fridge. Put the lettuce on the second shelf of the fridge, close the fridge, open the fridge and pick it back up. Turn left, go halfway across the room and turn right toward the coffee maker. Put the slice of lettuce on the counter in front of the coffee maker.	turn left and go around the counter top, then go straight to the stove top. pick up the knife with the yellow handle from behind the salt shaker on the counter top. turn left and face the counter top to your left. slice the lettuce on the counter top. face the counter top with the knife in hand. place the knife down next to the lettuce slices on the counter top. face the lettuce on the counter top. pick up a slice of lettuce on the counter top. turn left, then face the opposite wall behind you to face the refrigerator. open the refrigerator, and place the slice of lettuce in front of the apple one shelf above the bread, then shut the door and open it up again after several seconds to pick the lettuce slice up. turn left, then face forward to the part of the counter top on which the coffee maker sits. place the slice of lettuce in front of the coffee maker.

Examine in Light

	Annotation # 1	Annotation # 2	Annotation # 3
Goals	**Read a book by lamp light.**	**Examine a book with a lamp.**	**Pick up a book and turn on a lamp.**
Instructions	Head forward to the bed in front of you. Pick up the blue book that is sitting on the bed; the book that says Probabilistic Robotics. Turn to your right and walk to the night stand. Turn on the lamp that is sitting on the night stand.	walk forward a few steps, turn right, take two steps, turn left, walk to bed. pick up the book that is on the bed. turn around, take a step, turn left to face small table. turn the lamp on.	Walk forward to face the bed. Pick the book up from the bed. Turn to the right and face the night stand with the lamp. Turn the lamp on.

FIGURE A.9: **Dataset Examples – Part 2.** Annotations for seven expert demonstrations.

Appendix B

CLIPort

B.1 Task Details

We extend the Ravens benchmark [311] to 10 language-conditioned. 8 out of 10 tasks have two evaluation variants, denoted by seen and unseen in their names. All tasks use hand-coded experts to generate expert demonstrations. These experts use privileged state information from the simulator along with pre-specified heuristics to complete the tasks. We refer the reader to the original Transporter paper [311] for details regarding these experts. The following is a description of each language-conditioned task:

B.1.1 Align Rope

Example: Figure 3.1(a).
Task: Manipulate a deformable rope to connect its end-points between two corners of a 3-sided square. There are four possible combinations for aligning the rope: "front left tip to front right tip". "front right tip to back right corner", "front left tip to back left corner", and "back right corner to back left corner". Here 'front' and 'back' refer to canonical positions on the 3-sided square. The poses of both the rope and 3-sided square are randomized for each task instance.
Objects: All align-rope instances contain a rope with 20 articulated beads and a 3-sided square.
Success Metric: The poses of all beads match the line segments between the two correct sides.

B.1.2 Packing Unseen Shapes

Example: Figure 3.1(b).
Task: Place a specified shape in the brown box. Each task instance contains 1 shape to be picked along with 4 distractor shapes. The shape colors are randomized but have no relevance to the task. This task does not require precise placements and is mostly a test of the agent's semantic understanding of arbitrary shapes.
Objects: packing-unseen-shapes is trained with seen shapes but evaluated on unseen shapes from Figure 3.4.
Success Metric: The correct shape is inside the bounds of the brown box.

B.1.3 Assembling Kits Seq

Example: Figure 3.1(c).
Task: Precisely place each specified shape in the specified hole following the order prescribed in the language instruction at each timestep. This is one of the hardest tasks in the benchmark requiring precise placements of unseen shapes of unseen colors and grounding spatial relationships like "the middle square hole" or "the bottom letter R hole". Each task instance contains 5 shapes and a kit with randomized poses.
Objects: Both assembling-kits-seq-seen-colors and assembling-kits-seq-unseen-colors are trained on seen shapes but evaluated on unseen shapes from Figure 3.4. However for color randomization, assembling-kits-seq-seen-colors is trained and evaluated on seen colors, and assembling-kits-seq-unseen-colors is trained with seen colors but evaluated on unseen colors from Figure 3.4.
Success Metric: The pose of each shape matches the specified hole at the correct timestep. The final score is the total number of shapes that were placed in the correct pose at the correct timestep, divided by the total number of shapes in the scene (always 5).

B.1.4 Put Blocks in Bowl

Example: Figure 3.1(d).
Task: Place all blocks of a specified color in a bowl of specified color. Each bowl fits just one block and all scenes contain enough bowls achieve the goal. Each task instance contains several distractor blocks and bowls with randomized colors. The solutions to this task are multimodal in that there could be several ways to place the blocks specified in the language goal. This task does not require precise placements and mostly tests an agent's ability to ground color attributes.
Objects: put-blocks-in-bowl-seen-colors is trained and evaluated on seen colors from Figure 3.4 for both blocks and bowls. put-blocks-in-bowl-unseen-colors is trained on seen colors but evaluated on unseen colors from Figure 3.4 for both blocks and bowls.
Success Metric: All blocks of the specified color are within the bounds a bowl of the specified color. The final score is the total number of correct blocks in the correct bowls, divided by the total number of relevant color blocks in the scene.

B.1.5 Packing Box Pairs

Example: Figure 3.1(e).
Task: Tightly pack all the boxes of two specified colors inside the brown box. All scenes contain the exact number of relevant color blocks to fill the box completely, but also contain some distractor boxes of irrelevant colors. The sizes of the boxes and the brown box are randomized. The distractor objects have equivalent sizes to the relevant objects to make the task more difficult. Sometimes the scene only contains one of the two specified specified colors and the agent has to actively ignore the missing color. Overall, this task requires both semantic understanding of colors and precise spatial reasoning for tightly packing boxes of unknown sizes.

Objects: Boxes with randomized widths and lengths and a brown box. packing-box-pairs-seen-colors is trained and evaluated on seen color boxes from Figure 3.4. packing-box-pairs-unseen-colors is trained on seen color boxes but evaluated on unseen color boxes from Figure 3.4.
Success Metric: All blocks of the two specified colors are tightly packed inside the bounds of the brown box. The final score is the total volume of the correct color blocks inside the box, divided by the total volume of the relevant color blocks in the scene.

B.1.6 Packing Google Objects Seq

Example: Figure 3.1(f).
Task: Place the specified objects in the brown box following the order prescribed in the language instruction at each timestep. This task does not require precise placements and mostly evaluates an agent's ability to ground semantic object descriptions. All objects in a scene are unique without any duplicates. The poses of the objects and the box are randomized for each scene.
Objects: packing-seen-google-objects-seq is trained and evaluated on all 56 objects in Figure 3.4. packing-unseen-google-objects-seq is trained on 37 seen objects but evaluated on 19 unseen objects in Figure 3.4.
Success Metric: Each specified object is within the bounds of the brown box at the correct timestep. The final score is the total volume of the correct objects placed inside the box at the correct timestep, divided by the total volume of the relevant objects.

B.1.7 Packing Google Objects Group

Example: Figure 3.1(g).
Task: Place all objects of the specified category in the brown box. This task does not require precise placements or following a specific action sequence. Each scene contains objects of multiple categories with each category containing at least 2 duplicates. The task cannot be solved by counting the number of objects since there are distractor objects, each with 2 or more duplicates.
Objects: packing-seen-google-objects-group is trained and evaluated on all 56 objects in Figure 3.4. packing-unseen-google-objects-group is trained on 37 seen objects but evaluated on 19 unseen objects in Figure 3.4.
Success Metric: All specified objects of a category are within the bounds of the brown box. The final score is the total volume of the correct objects in the box, divided by the total volume of the relevant objects of the specified category in the scene.

B.1.8 Stack Block Pyramid

Example: Figure 3.1(h).
Task: Build a pyramid of colored blocks in a color sequence specified through the step-by-step language instructions. Each task contains 6 blocks with randomized colors and 1 rectangular base, all initially placed at random poses.

Objects: 6 blocks and 1 rectangular base. stack-block-pyramid-seq-seen-colors is trained and evaluated on seen color blocks from Figure 3.4. stack-block-pyramid-seq-unseen-colors is trained on seen color blocks but evaluated on unseen color blocks from Figure 3.4.
Success Metric: The pose of each block at the corresponding timestep matches the specified location. The final score is the total number of blocks in the correct pose at the correct timestep, divided by the total number of blocks (always 6).

B.1.9 Separating Piles

Example: Figure 3.1(i).
Task: Sweep the pile of blocks into the specified zone. Each scene contains two square zones: one relevant to the task, another as a distractor. The pile and zones are placed at random poses on the table.
Objects: A pile of colored blocks and two squares. separating-piles-seen-colors is trained and evaluated on seen colors from Figure 3.4 for all blocks and squares. separating-piles-unseen-colors is trained on seen colors but evaluated on unseen colors from Figure 3.4 for all blocks and squares.
Success Metric: All blocks are inside the bounds of the specified zone. The final score is the total number of blocks inside the correct zone, divided by the total number of blocks in the scene.

B.1.10 Towers of Hanoi Seq

Example: Figure 3.1(j).
Task: Move the ring to the specified peg in the language instruction at each timestep. The sequence of ring placements is always the same, i.e. the perfect solution to three-ring Towers of Hanoi. This task can be solved without using colors by just observing the ring sizes. However, it tests the agent's ability to ignore irrelevant concepts to the task (color in this case). The task involves precise pick and place actions for moving the rings from peg to peg.
Objects: 1 peg base and 3 rings (small, medium, and big). towers-of-hanoi-seen-colors is trained and evaluated on seen ring colors from Figure 3.4. towers-of-hanoi-unseen-colors is trained on seen ring colors but evaluated on unseen ring colors from Figure 3.4.
Success Metric: The pose of each ring at the corresponding timestep matches the specified peg location. The final score is the total number of correct ring placements, divided by total steps in the perfect solution (7 for three-ring Towers of Hanoi).

Method	packing-box-pairs seen-colors				packing-box-pairs unseen-colors				packing-seen-google objects-seq				packing-unseen-google objects-seq				packing-seen-google objects-group				packing-unseen-google objects-group			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [311]	48.9	57.2	59.4	60.6	37.8	52.3	54.5	60.7	30.2	41.6	42.4	46.3	26.3	37.1	42.9	40.8	56.3	52.8	55.6	54.5	30.8	55.3	53.6	56.0
CLIP-only	37.1	72.3	87.4	90.9	36.1	61.8	67.2	62.9	30.5	76.5	89.1	**97.7**	37.8	48.9	55.2	58.9	53.3	66.1	90.6	94.6	46.7	63.3	76.7	78.1
RN50-BERT	40.0	64.4	94.7	90.5	42.1	58.7	62.4	72.2	29.7	49.8	90.4	94.6	39.9	41.8	57.5	57.2	48.5	56.9	83.1	93.6	44.8	55.3	71.7	77.9
CLIPort (single)	51.9	84.7	95.9	**98.0**	47.1	66.9	70.0	71.9	14.4	63.9	**95.3**	96.9	25.0	50.6	62.7	62.0	53.3	72.5	90.3	**95.6**	54.9	68.5	78.3	73.3
CLIPort (multi)	**68.6**	**90.0**	**96.0**	96.3	**55.9**	**70.3**	**76.6**	**72.9**	**45.7**	**78.4**	83.8	83.4	**50.8**	**60.8**	**65.1**	**68.8**	**69.4**	**86.2**	**92.2**	93.2	**66.9**	**73.4**	**82.0**	**81.7**
CLIPort (multi-attr)	–	–	–	–	*46.2*	*72.0*	*86.2*	*80.3*	–	–	–	–	*35.4*	*45.1*	*78.7*	*87.4*	–	–	–	–	*48.6*	*69.3*	*84.8*	*89.1*

Method	stack-block-pyramid seq-seen-colors				stack-block-pyramid seq-unseen-colors				separating-piles seen-colors				separating-piles unseen-colors				towers-of-hanoi seq-seen-colors				towers-of-hanoi seq-unseen-colors			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [311]	4.8	4.0	6.8	5.7	4.8	5.3	5.0	5.0	42.8	52.9	54.7	55.6	47.8	53.4	52.6	54.8	25.1	74.4	**100**	**100**	25.6	46.4	77.0	81.7
CLIP-only	5.5	30.0	58.7	59.0	2.0	16.3	5.7	19.3	39.7	69.6	90.4	92.9	46.4	61.6	76.9	74.4	10.9	48.1	88.6	52.9	15.9	44.7	67.1	58.1
RN50-BERT	5.7	35.5	94.0	98.0	5.2	10.5	19.7	33.3	33.3	55.9	53.0	48.7	35.7	52.2	53.1	57.0	26.4	68.1	92.7	95.9	16.3	75.0	82.0	84.3
CLIPort (single)	29.0	68.8	95.0	**99.3**	15.8	29.0	32.7	**41.8**	45.1	58.6	**96.8**	**99.9**	50.7	56.5	**83.8**	**83.0**	55.3	**94.1**	99.9	**100**	**66.6**	**91.9**	**96.4**	**100**
CLIPort (multi)	**38.3**	**71.0**	**97.0**	97.3	**27.8**	**31.8**	**39.3**	33.3	**53.2**	**73.0**	92.7	89.2	**55.5**	**71.2**	79.5	76.7	**67.6**	94.0	99.1	**100**	55.6	68.6	79.1	67.0
CLIPort (multi-attr)	–	–	–	–	*17.2*	*45.2*	*65.3*	*81.5*	–	–	–	–	*49.9*	*51.8*	*48.2*	*59.8*	–	–	–	–	*56.7*	*78.0*	*88.3*	*96.9*

Method	align-rope				packing-unseen-shapes				assembling-kits-seq seen-colors				assembling-kits-seq unseen-colors				put-blocks-in-bowls seen-colors				put-blocks-in-bowls unseen-colors			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [311]	6.3	24.7	39.8	48.2	28.0	34.0	27.0	32.0	6.8	15.2	30.8	32.6	9.4	15.6	30.4	30.0	18.8	45.2	63.2	69.0	12.2	16.8	20.5	21.7
CLIP-only	15.4	47.6	76.7	74.3	26.0	36.0	40.0	**43.0**	1.4	6.4	19.0	27.2	4.2	5.6	12.0	16.2	22.3	62.2	94.7	98.5	15.8	29.7	38.3	24.7
RN50-BERT	6.8	26.9	69.8	61.1	22.0	31.0	29.0	30.0	2.4	6.8	15.2	23.0	2.2	7.6	15.2	19.4	10.8	46.3	82.3	92.2	14.0	24.2	29.7	27.7
CLIPort (single)	14.8	**66.2**	**93.2**	**98.2**	22.0	**42.0**	35.0	40.0	11.0	28.8	**51.6**	**72.0**	**17.2**	23.2	33.0	**38.0**	21.7	73.0	98.2	**100**	17.2	32.5	40.2	**48.3**
CLIPort (multi)	**19.2**	52.4	80.2	72.2	**29.0**	**42.0**	**47.0**	41.0	**17.4**	**37.2**	48.2	57.6	12.2	**23.8**	**36.4**	29.0	**59.7**	**94.0**	**100**	**100**	**33.8**	**42.7**	**55.3**	43.3
CLIPort (multi-attr)	–	–	–	–	–	–	–	–	–	–	–	–	*9.0*	*18.4*	*41.6*	*39.8*	–	–	–	–	*23.0*	*41.8*	*66.5*	*75.7*

TABLE B.1: **Validation Results.** Task success scores (mean %) from 100 evaluation instances vs. # of training demonstrations (1, 10, 100, or 1000). The challenges pertaining to each task can be found in Appendix B.1. CLIPort (single) models are trained on seen splits, and evaluated on both seen and unseen splits. CLIPort (multi) models are trained on seen splits of all 10 tasks with 1$\mathbb{T}$, 10$\mathbb{T}$, 100$\mathbb{T}$, and 1000$\mathbb{T}$ demonstrations where $\mathbb{T} = 10$. CLIPort (multi-attr) indicate CLIPort (multi) models trained on seen-and-unseen splits from all tasks *except* for that one particular heldout task, for which it is trained only the seen split. See Figure B.1 for an overview with average scores.

B.2 Evaluation Worflow and Validation Results

Evaluation Workflow. All simulated experiments in Section 3.4.2 follow a four-phase workflow: (1) generate train, validation, and test sets, (2) train agents on the train set, (3) optimize on the validation set to find the best checkpoint, (4) evaluate the best checkpoint on the test set. Both validation and test sets consist of 100 evaluation instances each. We found that validation loss is a poor metric for determining the best checkpoint as actions are often multi-modal. In a task like "put the yellow blocks in the red bowl" where there are three possible yellow blocks to choose from, the validation loss is high if the agent chooses a different yellow block to the expert, but in fact choosing any yellow block would suffice in achieving the goal. This issue is addressed by determining the best checkpoint through task execution performance on the validation set.

Validation Performances. During validation, we evaluate a trained agent across fixed checkpoints between 1K-200K iterations for single-task settings and 1K-600K iterations for multi-task settings. We then choose the best-performing checkpoint for each task. Table B.1 presents validation results for all tests in Section 3.4.2. Following Transporter [311], we use a learning rate of `1e-4` with no additional hyperparameter tuning. We note that better learning rate

schedules and other hyperparameter optimizations could possibly improve the performance of agents, especially in multi-task settings.

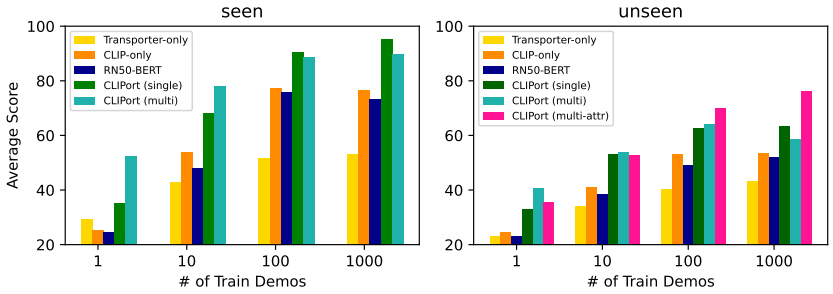


FIGURE B.1: Average validation scores across seen and unseen splits for all tasks in Table B.1.

B.3 Robot Setup

FIGURE B.2: **Real-Robot Experimental Setup.**

Hardware Setup. All real-robot experiments were conducted on a Franka Panda robot with a parallel-gripper. For perception, we use a Kinect-2 RGB-D camera mounted on a tripod, tilted down looking at the table. Although the Kinect-2 provides images at a resolution of 1280×720, we use downsampled 960×540 images for a faster user-interface. The extrinsic calibration between the camera and the robot base-frame is computed with an AR Marker through ARUCO ROS[1]. See Figure B.2 for an overview of the setup.

Demonstrations and Execution. For collecting demonstrations with the Franka Panda, we developed a 2D interactive tool that uses the top-down RGB view from the Kinect-2 to specify pick-and-place locations. The user first selects a 2D bounding box on the live RGB feed, and then picks a discrete rotation angle by clicking around the bounding box. For grasping, we use a simple heuristic to determine the height at which to close the fingers. First we segment the pointcloud encapsulated by the bounding box, then we vertically crop the pointcloud up to the height of the gripper fingers, and then compute a 3D centroid of the selected points by taking an average. This 3D centroid is used to plan a path for the end-effector with an RRT* motion-planner to execute a predefined sequence –

[1] https://github.com/pal-robotics/aruco_ros

go down, open/close the gripper, raise up. For executing a trained CLIPort model, a similar grasping approach is used, but instead of the user-specified bounding box, we take 32×32 crops centered around the pick and place predictions (i.e. affordance argmax) to compute 3D centroids from the pointcloud. Only the sweeping and folding actions are different in that the end-effector does not raise up after grasping.

Pick Rotations for Parallel Grippers. The suction gripper used in simulation does not require a pick rotation since the grasps are specified as pin-point locations. However, with the Franka Panda, the parallel gripper requires a specific yaw rotation at which to grasp an object. To handle this, we separate the pick module $\mathcal{Q}_{\text{pick}}$ into two components: locator and rotator. The locator predicts a pixel location (u, v) given the full observation and language input. The rotator takes a 64×64 crop of the observation at (u, v) along with the language input and predicts a discrete rotation angle by selecting from one of k rotated crops. We use $k = 36$ in all our hardware experiments. While it's possible to predict both the location and rotation with a single module, this decoupled approach allows us to fit the model on a single GPU (NVIDIA P100) with reduced memory usage from cropped rotations.

B.4 Two Stream Architecture Details

Figure B.3 provides a detailed architecture diagram of CLIPort's two-stream design. We use ReLU activations after each `conv` and `identity` blocks without any Batch Normalization. Note that we repeat the depth input to match the dimensions of the RGB image $\mathbb{R}^{H\times W\times 1} \rightarrow \mathbb{R}^{H\times W\times 3}$ following Transporter [311]. All models were implemented in PyTorch [216]. For CLIP, we use the implementation and pre-trained checkpoint released by the authors[2].

[2] https://github.com/openai/CLIP

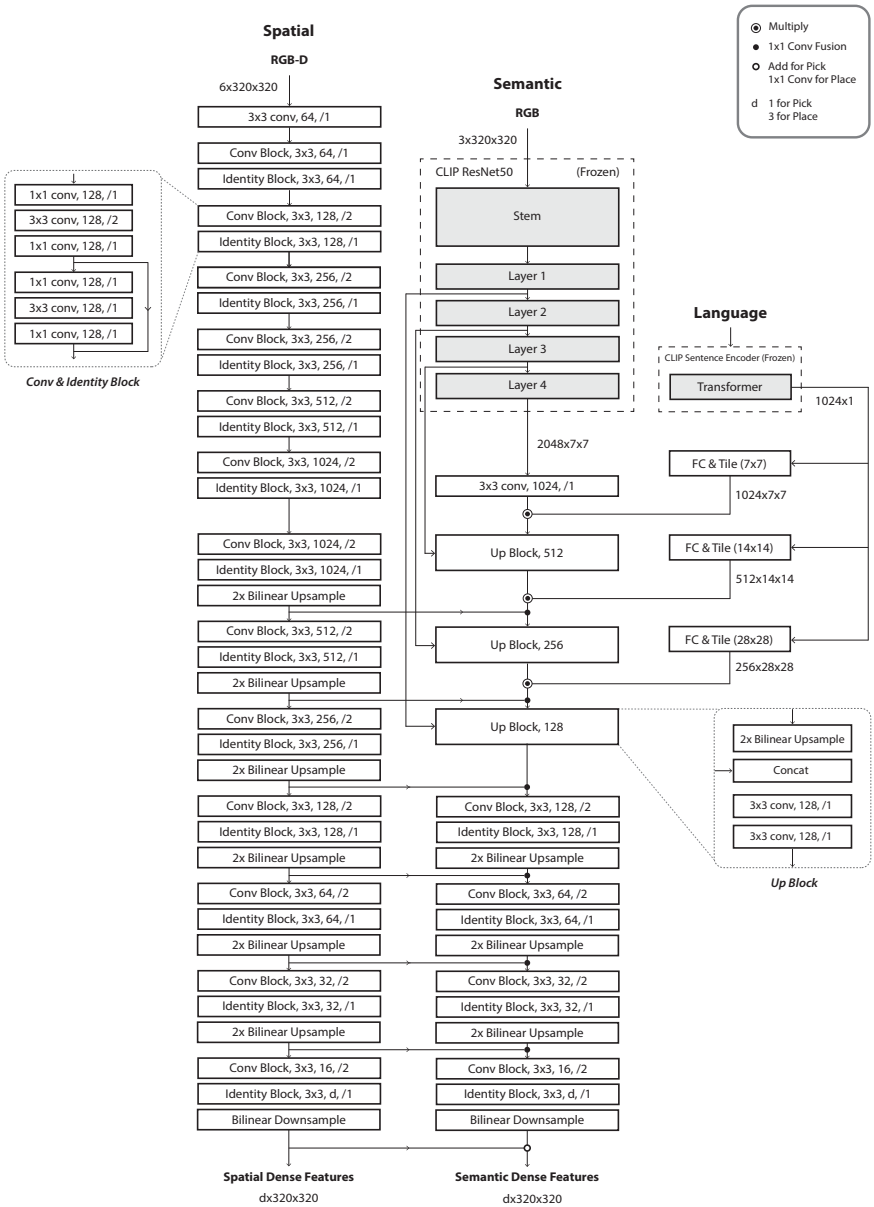


FIGURE B.3: **CLIPort Two-Stream Architecture.** A detailed architecture diagram of the semantic and spatial pathways.

B.5 Data Augmentation

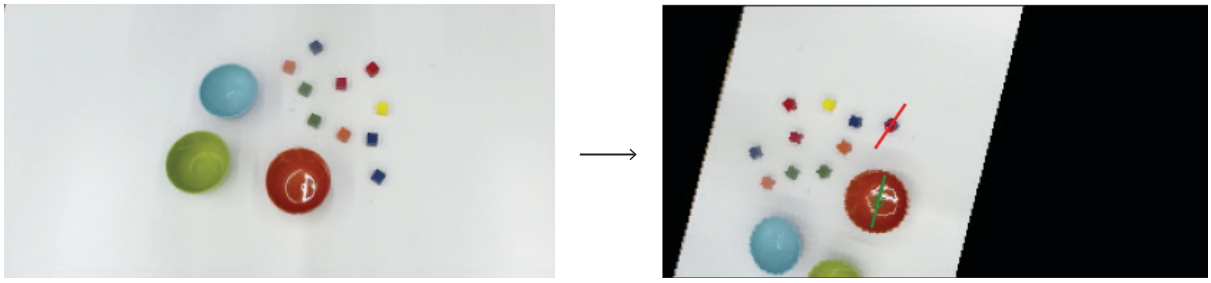


FIGURE B.4: **Data Augmentation.** $\mathbf{SE}(2)$ transform applied to RGB-D input. The left image shows the original input, and the right image shows the transformed input along with expert $\mathcal{T}_{\text{pick}}$ (red) and $\mathcal{T}_{\text{place}}$ (green) actions.

Following common practice and the original Transporter implementation [311], we augment the training samples by applying random $\mathbf{SE}(2)$ transformations. Augmentations where $\mathcal{T}_{\text{pick}}$ or $\mathcal{T}_{\text{place}}$ are out of frame after the transformation are discarded. These augmentations are particular important for learning spatially-equivariant representations with FCNs without overfitting to images from limited training demonstrations.

B.6 Affordance Prediction Examples

Figure B.5 showcases more examples of affordance predictions from trained CLIPort (multi) models. Traditional object-centric representations like pose and instance segmentation generally struggle to represent piles of beans or squares on a chessboard. In such cases, a single detector would have to be trained (with supervision data) to detect every bean and square on the chessboard, which is often infeasible, especially in multi-task settings.

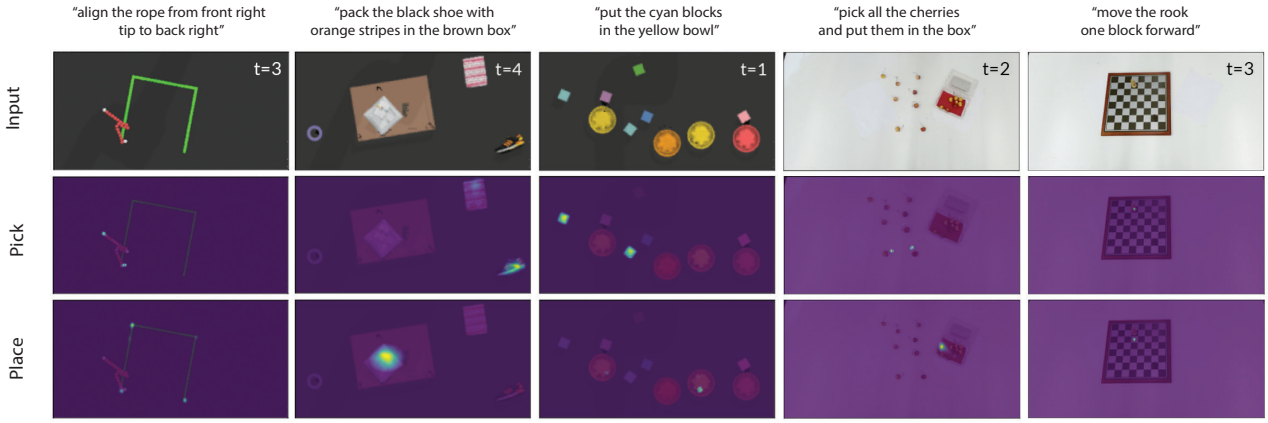


FIGURE B.5: More examples of pick and place affordance predictions from CLIPort (multi). The left three columns are from simulated tasks, and the right two columns are from real-world tasks.

Appendix C

PERACT

C.1 Task Details

Setup. Our simulated experiments are set in RLBench [127]. We select 18 out of 100 tasks that involve at least two or more variations to evaluate the multi-task capabilities of agents. While PERACT could be easily applied to more RLBench tasks, in our experiments, we were specifically interested grounding diverse language instructions, rather than learning one-off policies for single-variation tasks like *"[always] take off the saucepan lid"*. Some tasks were modified to include additional variations. See Table 4.1 for an overview. We report average keyframes extracted from the method described in Section 4.3.2.

Variations. Task variations include randomly sampled colors, sizes, shapes, counts, placements, and categories of objects. The set of colors include 20 instances: `colors` = {`red`, `maroon`, `lime`, `green`, `blue`, `navy`, `yellow`, `cyan`, `magenta`, `silver`, `gray`, `orange`, `olive`, `purple`, `teal`, `azure`, `violet`, `rose`, `black`, `white`}. The set of sizes include 2 instances: `sizes` = {`short`, `tall`}. The set of shapes include 5 instances: `shapes` = {`cube`, `cylinder`, `triangle`, `star`, `moon`}. The set of counts include 3 instances: `counts` = {1, 2, 3}. The placements and object categories are specific to each task. For instance, `open drawer` has 3 placement locations: `top`, `middle`, and `bottom`, and `put in cupboard` includes 9 YCB objects. In addition to these semantic variations, objects are placed on the tabletop at random poses. Some large objects like drawers have constrained pose variations [127] to ensure that manipulating them is kinematically feasible with the Franka arm.

In the following sections, we describe each of 18 tasks in detail. We highlight tasks that were modified from the original RLBench [127] codebase[1] and describe what exactly was modified:

C.1.1 Open Drawer

Filename: `open_drawer.py`
Task: Open one of the three drawers: `top`, `middle`, or `bottom`.
Modified: No.

[1] https://github.com/stepjam/RLBench

Objects: 1 drawer.
Success Metric: The prismatic joint of the specified drawer is fully extended.

C.1.2 Slide Block

Filename: `slide_block_to_color_target.py`
Task: Slide the block on to one of the colored square targets. The target colors are limited to `red`, `blue`, `pink`, and `yellow`.
Modified: Yes. The original `slide_block_to_target.py` task contained only one target. Three other targets were added to make a total of 4 variations.
Objects: 1 block and 4 colored target squares.
Success Metric: Some part of the block is inside the specified target area.

C.1.3 Sweep to Dustpan

Filename: `sweep_to_dustpan_of_size.py`
Task: Sweep the dirt particles to either the short or tall dustpan.
Modified: Yes. The original `sweep_to_dustpan.py` task contained only one dustpan. One other dustpan was added to make a total of 2 variations.
Objects: 5 dirt particles and 2 dustpans.
Success Metric: All 5 dirt particles are inside the specified dustpan.

C.1.4 Meat Off Grill

Filename: `meat_off_grill.py`
Task: Take either the chicken or steak off the grill and put it on the side.
Modified: No.
Objects: 1 piece of chicken, 1 piece of steak, and 1 grill.
Success Metric: The specified meat is on the side, away from the grill.

C.1.5 Turn Tap

Filename: `turn_tap.py`
Task: Turn either the left or right handle of the tap. Left and right are defined with respect to the faucet orientation.
Modified: No.
Objects: 1 faucet with 2 handles.
Success Metric: The revolute joint of the specified handle is at least 90° off from the starting position.

C.1.6 Put in Drawer

Filename: `put_item_in_drawer.py`
Task: Put the block in one of the three drawers: `top`, `middle`, or `bottom`.
Modified: No.
Objects: 1 block and 1 drawer.
Success Metric: The block is inside the specified drawer.

C.1.7 Close Jar

Filename: `close_jar.py`
Task: Put the lid on the jar with the specified color and screw the lid in. The jar colors are sampled from the full set of 20 color instances.
Modified: No.
Objects: 1 block and 2 colored jars.
Success Metric: The lid is on top of the specified jar and the Franka gripper is not grasping anything.

C.1.8 Drag Stick

Filename: `reach_and_drag.py`
Task: Grab the stick and use it to drag the cube on to the specified colored target square. The target colors are sampled from the full set of 20 color instances.
Modified: Yes. The original `reach_and_drag.py` task contained only one target. Three other targets were added with randomized colors.
Objects: 1 block, 1 stick, and 4 colored target squares.
Success Metric: Some part of the block is inside the specified target area.

C.1.9 Stack Blocks

Filename: `stack_blocks.py`
Task: Stack N blocks of the specified color on the green platform. There are always 4 blocks of the specified color, and 4 distractor blocks of another color. The block colors are sampled from the full set of 20 color instances.
Modified: No.
Objects: 8 color blocks (4 are distractors), and 1 green platform.
Success Metric: N blocks are inside the area of the green platform.

C.1.10 Screw Bulb

Filename: `light_bulb_in.py`
Task: Pick up the light bulb from the specified holder, and screw it into the lamp stand. The

colors of holder are sampled from the full set of 20 color instances. There are always two holders in the scene – one specified and one distractor holder.
Modified: No.
Objects: 2 light bulbs, 2 holders, and 1 lamp stand.
Success Metric: The bulb from the specified holder is inside the lamp stand dock.

C.1.11 Put in Safe

Filename: `put_money_in_safe.py`
Task: Pick up the stack of money and put it inside the safe on the specified shelf. The shelf has three placement locations: `top`, `middle`, `bottom`.
Modified: No.
Objects: 1 stack of money, and 1 safe.
Success Metric: The stack of money is on the specified shelf inside the safe.

C.1.12 Place Wine

Filename: `place_wine_at_rack_location.py`
Task: Grab the wine bottle and put it on the wooden rack at one of the three specified locations: `left`, `middle`, `right`. The locations are defined with respect to the orientation of the wooden rack.
Modified: Yes. The original `stack_wine.py` task had only one placement location. Two other locations were added to make a total of 3 variations.
Objects: 1 wine bottle, and 1 wooden rack.
Success Metric: The wine bottle is at the specified placement location on the wooden rack.

C.1.13 Put in Cupboard

Filename: `put_groceries_in_cupboard.py`
Task: Grab the specified object and put it in the cupboard above. The scene always contains 9 YCB objects that are randomly placed on the tabletop.
Modified: No.
Objects: 9 YCB objects, and 1 cupboard (that hovers in the air like magic).
Success Metric: The specified object is inside the cupboard.

C.1.14 Sort Shape

Filename: `place_shape_in_shape_sorter.py`
Task: Pick up the specified shape and place it inside the correct hole in the sorter. There are always 4 distractor shapes, and 1 correct shape in the scene.
Modified: Yes. The sizes of the shapes and sorter were enlarged so that they are distinguishable in the RGB-D input.

Objects: 5 shapes, and 1 sorter.
Success Metric: The specified shape is inside the sorter.

C.1.15 Push Buttons

Filename: `push_buttons.py`
Task: Push the colored buttons in the specified sequence. The button colors are sampled from the full set of 20 color instances. There are always three buttons in scene.
Modified: No.
Objects: 3 buttons.
Success Metric: All the specified buttons were pressed.

C.1.16 Insert Peg

Filename: `insert_onto_square_peg.py`
Task: Pick up the square and put it on the specified color spoke. The spoke colors are sampled from the full set of 20 color instances.
Modified: No.
Objects: 1 square, and 1 spoke platform with three color spokes.
Success Metric: The square is on the specified spoke.

C.1.17 Stack Cups

Filename: `stack_cups.py`
Task: Stack all cups on top of the specified color cup. The cup colors are sampled from the full set of 20 color instances. The scene always contains three cups.
Modified: No.
Objects: 3 tall cups.
Success Metric: All other cups are inside the specified cup.

C.1.18 Place Cups

Filename: `place_cups.py`
Task: Place N cups on the cup holder. This is a very high precision task where the handle of the cup has to be exactly aligned with the spoke of the cup holder for the placement to succeed.
Modified: No.
Objects: 3 cups with handles, and 1 cup holder with three spokes.
Success Metric: N cups are on the cup holder, each on a separate spoke.

C.2 PERACT Details

In this section, we provide implementation details for PERACT. See this Colab tutorial for a PyTorch implementation.

C.2.1 Input Observation.

Following James et al. [127], our input voxel observation is a 100^3 voxel grid with 10 channels: $\mathbb{R}^{100\times100\times100\times10}$. The grid is constructed by fusing calibrated pointclouds with PyTorch's `scatter_` function[2]. The 10 channels are composed of: 3 RGB, 3 point, 1 occupancy, and 3 position index values. The RGB values are normalized to a zero-mean distribution. The point values are Cartesian coordinates in the robot's coordinate frame. The occupancy value indicates if a voxel is occupied or empty. The position index values represent the 3D location of the voxel with respect to the 100^3 grid. In addition to the voxel observation, the input also includes proprioception data with 4 scalar values: gripper open, left finger joint position, right finger joint position, and timestep (of the action sequence).

C.2.2 Input Language.

The language goals are encoded with CLIP's language encoder [224]. We use CLIP's tokenizer to preprocess the sentence, which always results in an input sequence of 77 tokens (with zero-padding). These tokens are encoded with the language encoder to produce a sequence of dimensions $\mathbb{R}^{77\times512}$.

C.2.3 Preprocessing.

The voxel grid is encoded with a 3D convolution layer with a 1×1 kernel to upsample the channel dimension from 10 to 64. Similarly, the proprioception data is encoded with a linear layer to upsample the input dimension from 4 to 64. The encoded voxel grid is split into 5^3 patches through a 3D convolution layer with a kernel-size and stride of 5, which results in a patch tensor of dimensions $\mathbb{R}^{20\times20\times20\times64}$. The proprioception features are tiled in 3D to match the dimensions of the patch tensor, and concattenated along the channel to form a tensor of dimensions $\mathbb{R}^{20\times20\times20\times128}$. This tensor is flattened into a sequence of dimensions $\mathbb{R}^{8000\times128}$. The language features are downsampled with a linear layer from 512 to 128 dimensions, and then appended to the tensor to form the final input sequence to the Perceiver Transformer, which of dimensions $\mathbb{R}^{8077\times128}$. We also add learned positional embeddings to the input sequence. These embeddings are represented with trainable `nn.Parameter`(s) in PyTorch.

C.2.4 Perceiver Transformer

PerceiverIO [120] is a latent-space Transformer that uses a small set of latent vectors to encode extremely long input sequences. See Figure 4.3 for an illustration of this process. Perceiver first computes cross-attention between the input sequence and the set of latent vectors of dimensions $\mathbb{R}^{2048\times512}$. These latents are randomly initialized and trained end-to-end. The latents are encoded with 6 self-attention layers, and then cross-attended with the input to output a sequence that matches the input-dimensions. This output is upsampled with a 3D convolution

[2] https://pytorch.org/docs/stable/generated/torch.Tensor.scatter_.html

layer and tri-linear upsampling to form a voxel feature grid with 64 channels: $\mathbb{R}^{100\times100\times100\times64}$. This feature grid is concatenated with the initial 64-dimensional feature grid from the processing stage as a skip connection to the encoding layers. Finally, a 3D convolution layer with a 1×1 kernel downsamples the channels from 128 back to 64 dimensions. Our implementation of Perceiver is based on an existing open-source repository[3].

C.2.5 Decoding.

For translation, the voxel feature grid is decoded with a 3D convolution layer with a 1×1 kernel to downsample the channel dimension from 64 to 1. This tensor is the translation $\mathcal{Q}$-function of dimensions $\mathbb{R}^{100\times100\times100\times1}$. For rotation, gripper open, and collision avoidance actions, the voxel feature grid is max-pooled along the 3D dimensions to form a vector of dimensions $\mathbb{R}^{1\times64}$. This vector is decoded with three independent linear layers to form the respective $\mathcal{Q}$-functions for rotation, gripper open, and collision avoidance. The rotation linear layer outputs logits of dimensions $\mathbb{R}^{216}$ (72 bins of 5 degree increments for each of the three axes). The gripper open and collide linear layers output logits of dimensions $\mathbb{R}^2$.

Our codebase is built on the ARM repository[4] by James et al. [127].

C.3 Evaluation Workflow

C.3.1 Simulation

Simulated experiments in Section 4.4.2 follow a four-phase workflow: (1) generate a dataset with train, validation, and test sets, each containing 100, 25, and 25 demonstrations, respectively. (2) Train an agent on the train set and save checkpoints at intervals of 10K iterations. (3) Evaluate all saved checkpoints on the validation set, and mark the best performing checkpoint. (4) Evaluate the best performing checkpoint on the test set. While this workflow follows a standard train-val-test paradigm from supervised learning, it is not the most feasible workflow for real-robot settings. With real-robots, collecting a validation set and evaluating all checkpoints could be very expensive.

C.3.2 Real-Robot

For real-robot experiments in Section 4.4.7, we simply pick the last checkpoint from training. We check if the agent has been sufficiently trained by visualizing $\mathcal{Q}$-predictions on training examples with swapped or modified language goals. While evaluating a trained agent, the agent keeps acting until a human user stops the execution. We also visualize the $\mathcal{Q}$-predictions live to ensure that the agent's upcoming action is safe to execute.

[3] https://github.com/lucidrains/perceiver-pytorch
[4] https://github.com/stepjam/ARM

C.4 Robot Setup

All simulated experiments use the four camera setup illustrated in Figure 4.5. The front, left shoulder, and right shoulder cameras, are static, but the wrist camera moves with the end-effector. We did not modify the default camera poses from RLBench [127]. These poses maximize coverage of the tabletop, while minimizing occlusions caused by the moving arm. The wrist camera in particular is able to provide high-resolution observations of small objects like handles.

C.4.1 Real-Robot

Hardware Setup. The real-robot experiments use a Franka Panda manipulator with a parallel-gripper. For perception, we use a Kinect-2 RGB-D camera mounted on a tripod, at an angle, pointing towards the tabletop. See Figure C.1 for reference. We tried setting-up multiple Kinects for multi-view observations, but we could not fix the interference issue caused by multiple Time-of-Flight sensors. The Kinect-2 provides RGB-D images of resolution 512×424 at 30Hz. The extrinsics between the camera and robot base-frame are calibrated with the `easy_handeye` package[5]. We use an ARUCO[6] AR marker mounted on the gripper to aid the calibration process.

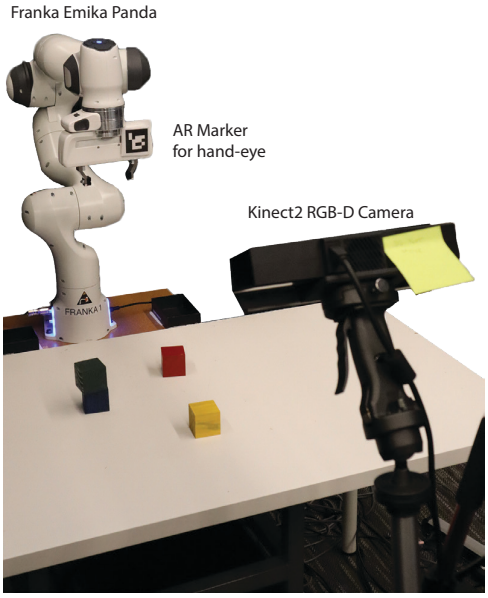


FIGURE C.1: **Real-Robot Setup** with Kinect-2 and Franka Panda.

Data Collection. We collect demonstrations with an HTC Vive controller. The controller is a 6-DoF tracker that provides accurate poses with respect to a static base-station. These poses are displayed as a marker on RViz[7] along with the real-time RGB-D pointcloud from the Kinect-2. A user specifies target poses by using the marker and pointcloud as reference. These target poses are executed with a motion-planner. We use Franka ROS and MoveIt[8], which by default uses an RRT-Connect planner.

Training and Execution. We train a PERACT agent from scratch with 53 demonstrations. The training samples are augmented with ± 0.125m translation perturbations and $\pm 45^{\circ}$ yaw rotation perturbations. We train on 8 NVIDIA P100 GPUs for 2 days. During evaluation, we

[5] https://github.com/IFL-CAMP/easy_handeye
[6] https://github.com/pal-robotics/aruco_ros
[7] http://wiki.ros.org/rviz
[8] http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/

simply chose the last checkpoint from training (since we did not collect a validation set for optimization). Inference is done on a single Titan X GPU.

C.5 Data Augmentation

PERACT's voxel-based formulation naturally allows for data augmentation with SE(3) transformations. During training, samples of voxelized observations $\mathbf{v}$ and their corresponding keyframe actions $\mathbf{k}$ are perturbed with random translations and rotations. Translation perturbations have a range of $[\pm 0.125\text{m}, \pm 0.125\text{m}, \pm 0.125\text{m}]$. Rotation perturbations are limited to the yaw axis and have a range of $[0°, 0°, \pm 45°]$. The $45°$ limit ensures that the perturbed rotations do not go beyond what is kinematically reachable for the Franka arm. We did experiment with pitch and roll perturbations, but they substantially lengthened the training time. Any perturbation that pushed the discretized action outside the observation voxel grid was discarded. See the bottom row of Figure 4.11 for examples of data augmentation.

C.6 Demo Augmentation

Following James et al. [127], we cast every datapoint in a demonstration as a "predict the next (best) keyframe action" task. See Figure C.2 for an illustration of this process. In this illustration, $\mathbf{k}_1$ and $\mathbf{k}_2$ are two keyframes that were extracted from the method described in Section 4.3.2. The orange circles indicate datapoints whose RGB-D observations are paired with the next keyframe action.

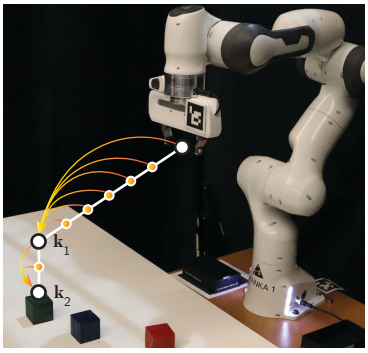


FIGURE C.2: **Keyframes and Demo Augmentation.**

C.7 Emergent Properties

In this section, we present some preliminary findings on the emergent properties of PERACT.

C.7.1 Object Tracking

Although PERACT was not explicitly trained for 6-DoF object-tracking, our action detection framework can be used to localize objects in cluttered scenes. In this video, we show an agent that was trained with one hand sanitizer instance on just 5 "press the handsan" demos, and then evaluated on tracking an unseen sanitizer instance. PERACT does not need to build a complete representation of hand sanitizers, and only has to learn *where to press* them. Our implementation runs at

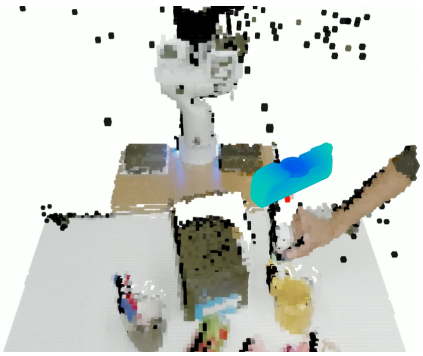


FIGURE C.3: **Object Tracker.** Tracking an unseen hand sanitizer instance.

an inference speed of 2.23 FPS (or 0.45 seconds per frame), allowing for near real-time closed-loop behaviors.

C.7.2 Multi-Modal Actions

PERACT's problem formulation allows for modeling multi-modal action distributions, i.e., scenarios where multiple actions are valid given a specific goal. Figure C.4 presents some selected examples of multi-modal action predictions from PERACT. Since there are several *"yellow blocks"* and *"cups"* to choose from, the Q-prediction distributions have several modes. In practice, we observe that the agent has a tendency to prefer certain object instances over others (like the front mug in Figure C.4) due to preference biases in the training dataset. We also note that the cross-entropy based training method from Section 4.3.4 is closely related to Energy-Based Models (EBMs) [159, 81]. In a way, the cross-entropy loss is *pulling up* expert 6-DoF actions, while *pushing-down* every other action in the discretized action space. At test time, we simply maximize the learned Q-predictions, instead of minimizing an energy function with optimization. Future works could look into EBM [81] training and inference methods for better generalization and execution performance.

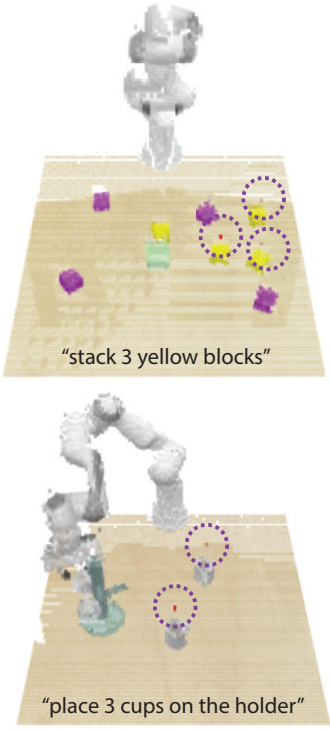


FIGURE C.4: **Examples of Multi-Modal Predictions.**

Appendix D

ALFWorld

D.1 Details of BUTLER::BRAIN

In this section, we use o_t to denote text observation at game step t, g to denote the goal description provided by a game. We use L to refer to a linear transformation and L^f means it is followed by a non-linear activation function f. Brackets $[\cdot;\cdot]$ denote vector concatenation, $\odot$ denotes element-wise multiplication.

D.1.1 Observation Queue

As mentioned in Section 5.4.1, we utilize an observation queue to cache the text observations that have been seen recently. Since the initial observation o_0 describes the high level layout of a room, including receptacles present in the current game, we it visible to BUTLER::BRAIN at all game steps, regardless of the length of the observation queue. Specifically, the observation queue has an extra space storing o_0, at any game step, we first concatenate all cached observations in the queue, then prepend the o_0 to form the input to the encoder. We find this helpful because it facilitates the pointer softmax mechanism in the decoder (described below) by guiding it to point to receptacle words in the observation. An ablation study on this is provided in Section 5.6.

D.1.2 Encoder

We use a transformer-based encoder, which consists of an embedding layer and a transformer block [281]. Specifically, embeddings are initialized by pre-trained 768-dimensional BERT embeddings [237]. The embeddings are fixed during training in all settings.

The transformer block consists of a stack of 5 convolutional layers, a self-attention layer, and a 2-layer MLP with a ReLU non-linear activation function in between. In the block, each convolutional layer has 64 filters, each kernel's size is 5. In the self-attention layer, we use a block hidden size H of 64, as well as a single head attention mechanism. Layernorm [20] is applied after each component inside the block. Following standard transformer training, we add positional encodings into each block's input.

At every game step t, we use the same encoder to process text observation o_t and goal description g. The resulting representations are $h_{o_t} \in \mathbb{R}^{L_{o_t} \times H}$ and $h_g \in \mathbb{R}^{L_g \times H}$, where L_{o_t} is the number of tokens in o_t, L_g denotes the number of tokens in g, $H = 64$ is the hidden size.

D.1.3 Aggregator

We adopt the context-query attention mechanism from the question answering literature [301] to aggregate the two representations h_{o_t} and h_g.

Specifically, a tri-linear similarity function is used to compute the similarity between each token in h_{o_t} with each token in h_g. The similarity between i-th token in h_o and j-th token in h_g is thus computed by (omitting game step t for simplicity):

$$\text{Sim}(i, j) = W(h_{o_i}, h_{g_j}, h_{o_i} \odot h_{g_j}), \tag{D.1}$$

where W is a trainable parameter in the tri-linear function. By applying the above computation for each h_o and h_g pair, we get a similarity matrix $S \in \mathbb{R}^{L_o \times L_g}$.

By computing the softmax of the similarity matrix S along both dimensions (number of tokens in goal description L_g and number of tokens in observation L_o), we get S_g and S_o, respectively. The two representations are then aggregated by:

$$\begin{aligned} h_{og} &= [h_o; P; h_o \odot P; h_o \odot Q], \\ P &= S_g h_g^\top, \\ Q &= S_g S_o^\top h_o^\top, \end{aligned} \tag{D.2}$$

where $h_{og} \in \mathbb{R}^{L_o \times 4H}$ is the aggregated observation representation.

Next, a linear transformation projects the aggregated representations to size $H = 64$:

$$h_{og} = L^{\tanh}(h_{og}). \tag{D.3}$$

To incorporate history, we use a recurrent neural network. Specifically, we use a GRU [52]:

$$\begin{aligned} h_{\text{RNN}} &= \text{Mean}(h_{og}), \\ h_t &= \text{GRU}(h_{\text{RNN}}, h_{t-1}), \end{aligned} \tag{D.4}$$

in which, the mean pooling is performed along the dimension of number of tokens, i.e., $h_{\text{RNN}} \in \mathbb{R}^H$. h_{t-1} is the output of the GRU cell at game step $t-1$.

D.1.4 Decoder

Our decoder consists of an embedding layer, a transformer block and a pointer softmax mechanism [96]. We first obtain the source representation by concatenating h_{og} and h_t, resulting $h_{\text{src}} \in \mathbb{R}^{L_o \times 2H}$.

Similar to the encoder, the embedding layer is frozen after initializing it with pre-trained BERT embeddings. The transformer block consists of two attention layers and a 3-layer MLP

with ReLU non-linear activation functions inbetween. The first attention layer computes the self attention of the input embeddings h_{self} as a contextual encoding for the target tokens. The second attention layer then computes the attention $\alpha^i_{\text{src}} \in \mathbb{R}^{L_o}$ between the source representation h_{src} and the i-th token in h_{self}. The i-th target token is consequently represented by the weighted sum of h_{src}, with the weights α^i_{src}. This generates a source information-aware target representation $h'_{\text{tgt}} \in \mathbb{R}^{L_{\text{tgt}} \times H}$, where L_{tgt} denotes the number of tokens in the target sequence. Next, h'_{tgt} is fed into the 3-layer MLP with ReLU activation functions inbetween, resulting $h_{\text{tgt}} \in \mathbb{R}^{L_{\text{tgt}} \times H}$. The block hidden size of this transformer is $H = 64$.

Taking h_{tgt} as input, a linear layer with tanh activation projects the target representation into the same space as the embeddings (with dimensionality of 768), then the pre-trained embedding matrix E generates output logits [222], where the output size is same as the vocabulary size. The resulting logits are then normalized by a softmax to generate a probability distribution over all tokens in vocabulary:

$$p_a(y^i) = E^{\text{Softmax}}(L^{\tanh}(h_{\text{tgt}})), \tag{D.5}$$

in which, $p_a(y^i)$ is the generation (abstractive) probability distribution.

We employ the pointer softmax [96] mechanism to switch between generating a token y^i (from a vocabulary) and pointing (to a token in the source text). Specifically, the pointer softmax module computes a scalar switch s^i at each generation time-step i and uses it to interpolate the abstractive distribution $p_a(y^i)$ over the vocabulary (Equation D.5) and the extractive distribution $p_x(y^i) = \alpha^i_{\text{src}}$ over the source text tokens:

$$p(y^i) = s^i \cdot p_a(y^i) + (1 - s^i) \cdot p_x(y^i), \tag{D.6}$$

where s^i is conditioned on both the attention-weighted source representation $\sum_j \alpha^{i,j}_{\text{src}} \cdot h^j_{\text{src}}$ and the decoder state h^i_{tgt}:

$$s^i = L_1^{\text{sigmoid}}(\tanh(L_2(\sum_j \alpha^{i,j}_{\text{src}} \cdot h^j_{\text{src}}) + L_3(h^i_{\text{tgt}}))). \tag{D.7}$$

In which, $L_1 \in \mathbb{R}^{H \times 1}$, $L_2 \in \mathbb{R}^{2H \times H}$ and $L_3 \in \mathbb{R}^{H \times H}$ are linear layers, $H = 64$.

D.2 Training and Implementation Details

In this section, we provide hyperparameters and other implementation details.

For all experiments, we use *Adam* [146] as the optimizer. The learning rate is set to 0.001 with a clip gradient norm of 5.

During training with DAgger, we use a batch size of 10 to collect transitions (tuples of $\{o_0, o_t, g, \hat{a}_t\}$) at each game step t, where $\hat{a}_t$ is the ground-truth action provided by the rule-based expert (see Section D.5). We gather a sequence of transitions from each game episode, and push each sequence into a replay buffer, which has a capacity of 500K episodes. We set the max number of steps per episode to be 50. If the agent uses up this budget, the game episode

is forced to terminate. We linearly anneal the fraction of the expert's assistance from 100% to 1% across a window of 50K episodes.

The agent is updated after every 5 steps of data collection. We sample a batch of 64 data points from the replay buffer. In the setting with the recurrent aggregator, every sampled data point is a sequence of 4 consecutive transitions. Following the training strategy used in the recurrent DQN literature [102, 309], we use the first 2 transitions to estimate the recurrent states, and the last 2 transitions for updating the model parameters.

BUTLER::BRAIN learns to generate actions token-by-token, where we set the max token length to be 20. The decoder stops generation either when it generates a special end-of-sentence token `[EOS]`, or hits the token length limit.

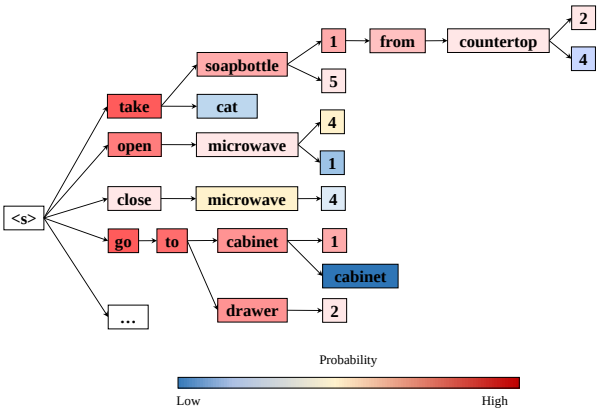


FIGURE D.1: Beam search for recovery actions.

When using the beam search heuristic to recover from failed actions (see Figure D.1), we use a beam width of 10, and take the top-5 ranked outputs as candidates. We iterate through the candidates in the rank order until one of them succeeds. This heuristic is not always guaranteed to succeed, however, we find it helpful in most cases. Note that we do not employ beam search when we evaluate during the training process for efficiency, e.g., in the seen and unseen curves shown in Figure 5.4. We take the best performing checkpoints and then apply this heuristic during evaluation and report the resulting scores in tables (e.g., Table 5.2).

By default unless mentioned otherwise (ablations), we use all available training games in each of the task types. We use an observation queue length of 5 and use a recurrent aggregator. The model is trained with DAgger, and during evaluation, we apply the beam search heuristic to produce the reported scores. All experiment settings in TextWorld are run with 8 random seeds. All text agents are trained for 50,000 episodes.

D.3 TextWorld Engine

Internally, the TextWorld Engine is divided into two main components: a planner and text generator.

Planner TextWorld Engine uses Fast Downward [108], a domain-independent classical planning system to maintain and update the current state of the game. A state is represented by a set of predicates which define the relations between the entities (objects, player, room, etc.) present in the game. A state can be modified by applying production rules corresponding to the actions listed in Table D.1. All variables, predicates, and rules are defined using the PDDL language.

For instance, here is a simple state representing a player standing next to a microwave which is closed and contains a mug:

$$\begin{aligned} s_t = \texttt{at}(\textit{player}, \textit{microwave}) \otimes \texttt{in}(\textit{mug}, \textit{microwave}) \\ \otimes \texttt{closed}(\textit{microwave}) \otimes \texttt{openable}(\textit{microwave}), \end{aligned} \tag{D.8}$$

where the symbol $\otimes$ is the linear logic *multiplicative conjunction* operator. Given that state, a valid action could be open *microwave*, which would essentially transform the state by replacing closed(*microwave*) with open(*microwave*).

Text generator The other component of the TextWorld Engine, the text generator, uses a context-sensitive grammar designed for the ALFRED environments. The grammar consists of text templates similar to those listed in Table D.1. When needed, the engine will sample a template given some context, i.e., the current state and the last action. Then, the template gets realized using the predicates found in the current state.

D.4 Mask R-CNN Detector

We use a Mask R-CNN detector [106] pre-trained on MSCOCO [163] and fine-tune it with additional labels from ALFRED training scenes. To generate additional labels, we replay the expert demonstrations from ALFRED and record ground-truth image and instance segmentation pairs from the simulator (AI2-THOR) after completing each high-level action e.g., goto, pickup etc. We generate a dataset of 50K images, and fine-tune the detector for 4 epochs with a batch size of 8 and a learning rate of 5*e*-4. The detector recognizes 73 object classes where each class could vary up to 1-10 instances. Since demonstrations in the kitchen are often longer as they involve complex sequences like heating, cleaning etc., the labels are slightly skewed towards kitchen objects. To counter this, we balance the number of images sampled from each room (kitchen, bedroom, livingroom, bathroom) so the distribution of object categories is uniform across the dataset.

D.5 Rule-based Expert

To train text agents in an imitation learning (IL) setting, we use a rule-based expert for supervision. A given task is decomposed into sequence of subgoals (e.g., for heat & place: find the object, pick the object, find the microwave, heat the object with the microwave, find the receptacle, place the object in the receptacle), and a closed-loop controller tries to sequentially execute these goals. We note that while designing rule-based experts for ALFWorld is relatively straightforward, experts operating directly in embodied settings like the PDDL planner used in ALFRED are prone to failures due to physical infeasibilities and non-deterministic behavior in physics-based environments.

D.6 Benefits of Training in TextWorld over Embodied World

Pre-training in TextWorld offers several benefits over directly training in embodied environments. Figure D.2 presents the performance of an expert (that agents are trained to imitate) across various environments. The abstract textual space leads to higher goal success rates resulting from successful navigation and manipulation subroutines. TextWorld agents also do not suffer from object mis-detections and slow execution speed.

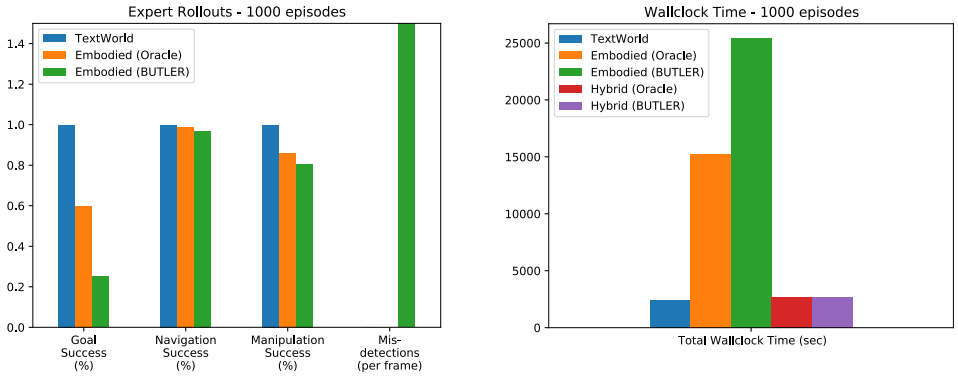


FIGURE D.2: **Domain Analysis.** The performance of an expert across various environments.

D.7 Observation Templates

The following templates are used by the state-estimator to generate textual observations o_t. The object IDs `{obj id}` correspond to Mask R-CNN objects detection or ground-truth instance IDs. The receptacle IDs `{recep id}` are based on the receptacles listed in the initial observation o_0. Failed actions and actions without any state-changes result in `Nothing happens`.

Actions	Templates
goto	(a) You arrive at {loc id}. On the {recep id}, you see a {obj1 id}, ... and a {objN id}. (b) You arrive at {loc id}. The {recep id} is closed. (c) You arrive at {loc id}. The {recep id} is open. On it, you see a {obj1 id}, ... and a {objN id}.
take	You pick up the {obj id} from the {recep id}.
put	You put the {obj id} on the {recep id}.
open	(a) You open the {recep id}. In it, you see a {obj1 id}, ... and a {objN id}. (b) You open the {recep id}. The {recep id} is empty.
close	You close the {recep id}.
toggle	You turn the {obj id} on.
heat	You heat the {obj id} with the {recep id}.
cool	You cool the {obj id} with the {recep id}.
clean	You clean the {obj id} with the {recep id}.
inventory	(a) You are carrying: {obj id}. (b) You are not carrying anything.
examine	(a) On the {recep id}, you see a {obj1 id}, ... and a {objN id}. (b) This is a hot/cold/clean {obj}.

TABLE D.1: High-level text actions supported in ALFWorld along with their observation templates.

D.8 Goal Descriptions

D.8.1 Templated Goals

The goal instructions for training games are generated with following templates. Here `obj`, `recep`, `lamp` refer to object, receptacle, and lamp classes, respectively, that pertain to a particular task. For each task, the two corresponding templates are sampled with equal probability.

D.8.2 Human Annotated Goals

The human goal descriptions used during evaluation contain 66 unseen verbs and 189 unseen nouns with respect to the templated goal instructions used during training.

task-type	Templates
Pick & Place	(a) put a {obj} in {recep}. (b) put some {obj} on {recep}.
Examine in Light	(a) look at {obj} under the {lamp}. (b) examine the {obj} with the {lamp}.
Clean & Place	(a) put a clean {obj} in {recep}. (b) clean some {obj} and put it in {recep}.
Heat & Place	(a) put a hot {obj} in {recep}. (b) heat some {obj} and put it in {recep}.
Cool & Place	(a) put a cool {obj} in {recep}. (b) cool some {obj} and put it in {recep}.
Pick Two & Place	(a) put two {obj} in {recep}. (b) find two {obj} and put them {recep}.

TABLE D.2: Task-types and the corresponding goal description templates.

Unseen Verbs: acquire, arrange, can, carry, chill, choose, cleaning, clear, cook, cooked, cooled, dispose, done, drop, end, fill, filled, frying, garbage, gather, go, grab, handled, heated, heating, hold, holding, inspect, knock, left, lit, lock, microwave, microwaved, move, moving, pick, picking, place, placed, placing, putting, read, relocate, remove, retrieve, return, rinse, serve, set, soak, stand, standing, store, take, taken, throw, transfer, turn, turning, use, using, walk, warm, wash, washed.

Unseen Nouns: alarm, area, back, baisin, bar, bars, base, basin, bathroom, beat, bed, bedroom, bedside, bench, bin, books, bottle, bottles, bottom, box, boxes, bureau, burner, butter, can, canteen, card, cardboard, cards, cars, cds, cell, chair, chcair, chest, chill, cistern, cleaning, clock, clocks, coffee, container, containers, control, controllers, controls, cooker, corner, couch, count, counter, cover, cream, credit, cupboard, dining, disc, discs, dishwasher, disks, dispenser, door, drawers, dresser, edge, end, floor, food, foot, freezer, game, garbage, gas, glass, glasses, gold, grey, hand, head, holder, ice, inside, island, item, items, jars, keys, kitchen, knifes, knives, laddle, lamp, lap, left, lid, light, loaf, location, lotion, machine, magazine, maker, math, metal, microwaves, move, nail, newsletters, newspapers, night, nightstand, object, ottoman, oven, pans, paper, papers, pepper, phone, piece, pieces, pillows, place, polish, pot, pullout, pump, rack, rag, recycling, refrigerator, remote, remotes, right, rinse, roll, rolls, room, safe, salt, scoop, seat, sets, shaker, shakers, shelves, side, sink, sinks, skillet, soap, soaps, sofa, space, spatulas, sponge, spoon, spot, spout, spray, stand, stool, stove, supplies, table, tale, tank, television, textbooks, time, tissue, tissues, toaster, top, towel, trash, tray, tv, vanity, vases, vault, vegetable, wall, wash, washcloth, watches, water, window, wine.

D.9 Action Candidates vs Action Generation

BUTLER::BRAIN generates actions in a token-by-token fashion. Prior text-based agents typically use a list of candidate commands from the game engine [3] or populate a list of command templates [10]. We initially trained our agents with candidate commands from the TextWorld Engine, but they quickly ovefit without learning affordances, commonsense, or pre-conditions,

and had zero performance on embodied transfer. In the embodied setting, without access to a TextWorld Engine, it is difficult to generate candidate actions unless a set of heuristics is handcrafted with strong priors and commonsense knowledge. We also experimented with populating a list of command templates, but found this to be infeasible as some scenarios involved 1000s of populated actions per game step.

D.10 ALFRED Task Descriptions

The following descriptions describe the processes involved in each of six `task-types`:

- Pick & Place (e.g., "put a plate on the coffee table") - the agent must find an object of the desired type, pick it up, find the correct location to place it, and put it down there.
- Examine in Light (e.g., "examine a book under the lamp") - the agent must find an object of the desired type, locate and turn on a light source with the desired object in-hand.
- Clean & Place (e.g., "clean the knife and put in the drawer") - the agent must find an object of the desired type, pick it up, go to a sink or a basin, wash the object by turning on the faucet, then find the correct location to place it, and put it down there.
- Heat & Place (e.g., "heat a mug and put on the coffee table") - the agent must find an object of the desired type, pick it up, go to a microwave, heat the object turning on the microwave, then find the correct location to place it, and put it down there.
- Cool & Place (e.g., "put a cool bottle on the countertop") - the agent must find an object of the desired type, pick it up, go to a fridge, put the object inside the fridge and cool it, then find the correct location to place it, and put it down there.
- Pick Two & Place (e.g., "put two pencils in the drawer") - the agent must find an object of the desired type, pick it up, find the correct location to place it, put it down there, then look for another object of the desired type, pick it up, return to previous location, and put it down there with the other object.

D.11 ALFWorld Text Game Examples

Three examples of ALFWorld text game walkthroughs:

You are in the middle of a room. Looking quickly around you, you see a towelholder 1,
a toilet 1, a bathtubbasin 1, a drawer 4, a handtowelholder 2, a drawer 6, a drawer 1,
a countertop 1, a sinkbasin 1, a drawer 2, a drawer 3, a toiletpaperhanger 1, a drawer 5,
a handtowelholder 1, a towelholder 2, a sinkbasin 2, and a garbagecan 1.

Your task is to: put a clean cloth in bathtubbasin.

> go to drawer 5
You arrive at loc 9. The drawer 5 is closed.
> open drawer 5
You open the drawer 5. The drawer 5 is open. In it, you see a candle 1.
> close drawer 5
You close the drawer 5.
> go to countertop 1
You arrive at loc 25. On the countertop 1, you see a cloth 2, a soapbottle 1, a cloth 1,
and a candle 3.
> take cloth 1 from countertop 1
You pick up the cloth 1 from the countertop 1.
> go to sinkbasin 1
You arrive at loc 20. On the sinkbasin 1, you see nothing.
> clean cloth 1 with sinkbasin 1
You clean the cloth 1 using the sinkbasin 1.
> go to bathtubbasin 1
You arrive at loc 14. On the bathtubbasin 1, you see a soapbar 1.
> put cloth 1 in/on bathtubbasin 1
You won!

You are in the middle of a room. Looking quickly around you, you see a drawer 15, a drawer 19,
a drawer 7, a armchair 1, a drawer 3, a drawer 16, a drawer 2, a drawer 17, a sidetable 3,
a sidetable 5, a drawer 4, a drawer 20, a sofa 1, a drawer 13, a drawer 12, a drawer 9,
a drawer 21, a sidetable 4, a sidetable 2, a drawer 5, a drawer 11, a drawer 1, a garbagecan 1,
a dresser 1, a cabinet 1, a drawer 8, a drawer 18, a sidetable 1, a drawer 14, a drawer 6,
and a drawer 10.

Your task is to: find two remotecontrol and put them in armchair.

> go to sidetable 2
You arrive at loc 34. On the sidetable 2, you see a remotecontrol 1.
> take remotecontrol 1 from sidetable 2
You pick up the remotecontrol 1 from the sidetable 2.
> go to armchair 1
You arrive at loc 1. On the armchair 1, you see nothing.
> put remotecontrol 1 in/on armchair 1
You put the remotecontrol 1 in/on the armchair 1.
> go to dresser 1
You arrive at loc 18. On the dresser 1, you see a creditcard 1, a television 1, a laptop 1,
a book 1, a statue 2, and a cellphone 2.
> go to sidetable 5
You arrive at loc 17. On the sidetable 5, you see a box 1, and a creditcard 2.
> go to sofa 1
You arrive at loc 2. On the sofa 1, you see a newspaper 1, a pillow 1, and a remotecontrol 2.
> take remotecontrol 2 from sofa 1
You pick up the remotecontrol 2 from the sofa 1.
> go to armchair 1
You arrive at loc 1. On the armchair 1, you see a remotecontrol 1.
> put remotecontrol 2 in/on armchair 1
You won!

You are in the middle of a room. Looking quickly around you, you see a safe 1, a shelf 4, a drawer 2,
a bed 1, a drawer 1, a shelf 5, a shelf 2, a sidetable 2, a shelf 3, a drawer 3, a shelf 1, a sidetable 1,
a desk 1, and a garbagecan 1.

Your task is to: examine the alarmclock with the desklamp.

> go to desk 1
You arrive at loc 8. On the desk 1, you see a pen 1, a bowl 1, a alarmclock 2, a pencil 2, a pencil 3,
a creditcard 3, a book 1, a alarmclock 3, a keychain 3, and a book 2.
> take alarmclock 2 from desk 1
You pick up the alarmclock 2 from the desk 1.
> go to sidetable 2
You arrive at loc 1. On the sidetable 2, you see a desklamp 1, and a alarmclock 1.
> use desklamp 1
You won!