UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning Embodied AI Agents with Task Decomposition

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Zhiwei Jia

Committee in charge:

Professor Hao Su, Chair
Professor Henrik I. Christensen
Professor Zhuowen Tu
Professor Xiaolong Wang

2023

DEDICATION

To my parents who always support me, those important ones in my life who have guided me through the darkness, and finally myself.

EPIGRAPH

A year spent in artificial intelligence
is enough to make one believe in God

Alan Perlis

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my PhD advisor Hao Su for his guidance and support during the five years of my PhD study. He taught me important lessons about being a great AI researcher and pursuing academic excellence in general. Special thanks to my former advisor Zhuowen Tu who guided me to this fantastic world of AI when I was an undergraduate, passionate yet disoriented.

Thanks to my parents who always gave me unconditional support. In some of the darkest moments, I could only regroup myself knowing that someone carried the umbrella somewhere for me. I really love you!

Many thanks to my labmates who accompanied me throughout this painstaking journey, especially Tongzhou, Shuang, and Fangchen.

Thanks to those important ones whom I fell for, liked, and loved. The PhD journey wasn't very easy with rainy days here and there. Sadly, the same goes for the beautiful journeys with them. I am genuinely grateful for all those unforgettable memories and valuable time, which kept making me a better person.

Last but not least, I would like to thank a large group of people - my friends, and of course, special thanks to the few closest ones. I really couldn't have come this far without you guys, who have accompanied and supported me over the years, actively or passively. Hope I have brought enough support and joy in return and will be capable of doing better in the future. Stay connected and see you around!

Chapter 2, in full, is a reprint of the material published in the 2022 International Conference on Intelligent Robots and Systems (IROS): "Learning to Act with Affordance-Aware Multimodal Neural SLAM" (Zhiwei Jia, Kaixiang Lin, Yizhou Zhao, Qiaozi Gao, Govind Thattai, and Gaurav Sukhatme). The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the manuscript: "Chain-of-Thought Predictive Control" (Zhiwei Jia, Vineet Thumuluri, Fangchen Liu, Linghao Chen, Zhiao Huang, Hao

Su), whose preliminary version has been presented at the Workshop on Reincarnating Reinforcement Learning at ICLR 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of the material published in the 2022 International Conference on Machine Learning (ICML): “Improving Policy Optimization with Generalist-Specialist Learning” (Zhiwei Jia, Xuanlin Li, Zhan Ling, Shuang Liu, Yiran Wu, Hao Su). The dissertation author was the primary investigator and author of this paper.

Chapter 1 contains citations to other published materials during my PhD study of which I was the primary investigator and author.

VITA

2014–2017	Bachelor of Science in Computer Science and Applied Math, UC San Diego
2018–2023	Doctor of Philosophy in Computer Science, UC San Diego

PUBLICATIONS

[1] "Chain-of-Thought Predictive Control". Z. Jia, F. Liu, V. Thumuluri, L. Chen, Z. Huang, and H. Su. ICLR 2023 RRL Workshop (preliminary version)
[2] "KAFA: Rethinking Image Ad Understanding with Knowledge-Augmented Feature Adaptation of VLMs". Z. Jia, P. Narayana, A. Akula, G. Pruthi, H. Su, S. Basu, and V. Jampani. ACL 2023 (Industry Track)
[3] "MetaCLUE: Towards Comprehensive Visual Metaphors Research". A. Akula, B. Driscoll, P. Narayana, S. Changpinyo, Z. Jia, S. Damle, G. Pruthi, S. Basu, L. Guibas, W. Freeman, Y. Li, and V. Jampani. CVPR 2023
[4] "Improving Policy Optimization with Generalist-Specialist Learning". Z. Jia, X. Li, Z. Ling, S. Liu, Y. Wu, and H. Su. ICML 2022
[5] "Learning to Act with Affordance-Aware Multimodal Neural SLAM". Z. Jia, K. Lin, Y. Zhao, Q. Gao, G. Thattai, and G. Sukhatme. IROS 2022
[6] "LUMINOUS: Indoor Scene Generation for Embodied AI Challenges". Y. Zhao, K. Lin, Z. Jia, Q. Gao, G. Thattai, J. Thomason, and G. Sukhatme. NeurIPS 2022 CtrlGen Workshop
[7] "ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations". T. Mu, Z. Ling, F. Xiang, D. Yang, X. Li, S. Tao, Z. Huang, Z. Jia, and H. Su. NeurIPS 2021 (Dataset Track)
[8] "Semantically Robust Unpaired Image Translation for Data with Unmatched Semantics Statistics". Z. Jia, B. Yuan, K. Wang, H. Wu, D. Clifford, Z. Yuan, and H. Su. ICCV 2021
[9] "Refactoring Policy for Compositional Generalizability using Self-Supervised Object Proposals". T. Mu, J. Gu, Z. Jia, H. Tang, and H. Su. NeurIPS 2020
[10] "One-pixel Signature: Characterizing CNN Classifiers for Backdoor Detection". S. Huang, W. Peng, Z. Jia, and Z. Tu. ECCV 2020
[11] "Information-Theoretic Local Minima Characterization and Regularization". Z. Jia and H. Su. ICML 2020

ABSTRACT OF THE DISSERTATION

Learning Embodied AI Agents with Task Decomposition

by

Zhiwei Jia

Doctor of Philosophy in Computer Science

University of California San Diego, 2023

Professor Hao Su, Chair

One of the ultimate goals of artificial intelligence (AI) is to build autonomous agents that can perceive, reason, and interact with the surroundings (i.e., Embodied AI). Despite the recent advances in deep learning and simulators, acquiring embodied agents via robot learning remains extremely demanding. In this dissertation, we present three methods, each from a different aspect, that adopt task decomposition to improve robot learning.

Training agents that follow human instructions to complete long-horizon household tasks, especially solely from offline data, poses several technical challenges, such as scene understanding and compositionally generalizable task executions (usually high-level actions). We propose to decompose the tasks into exploration and execution phases. In the first phase, we utilize multimodal signals to explore the scene in a task-driven manner to obtain an affordance-aware semantic map. Next, we adopt a hierarchical task execution system to complete the sub-task sequences (another level of task decomposition) according to the map and the instructions.

Short-horizon tasks involving low-level motor controls are usually harder for AI (Moravec's paradox). To solve challenging contact-rich object manipulation that entails high precision and/or object variations, we develop a novel hierarchical imitation learning method that utilizes scalable, albeit sub-optimal, demonstrations by further decomposing short-horizon tasks into subskills. We first propose an observation space-agnostic method that unsupervisedly discovers the multi-step subskill decomposition (sequences of key observations) from demos. Next, we propose a Transformer-based design that effectively learns to dynamically predict such subskill decomposition as the high-level future guidance for low-level actions.

Besides the hierarchical principles, task decomposition also refers to decomposing the task space itself. While large-scale RL over diverse environment variations poses great optimization challenges, we find that launching a population of agents (the specialists), each trained on a subset of the task variations, drastically eases policy optimization. We, therefore, propose a meta-framework that generally improves online RL methods to tackle complex tasks by combining distributed and joint training in a principled manner. Our approach achieves a great balance of efficiency and effectiveness in large-scale policy learning, which is verified with extensive ablation studies and a diverse set of benchmark tasks.

Chapter 1

Introduction

In this chapter, we mainly introduce the background of Embodied AI, representative simulators, datasets, and tasks (as well as their major challenges) used in Embodied AI research. We then briefly discuss the general motivations behind the three approaches presented in this dissertation as well as the structure of the remaining chapters.

1.1 Background of Embodied AI

The history of Artificial Intelligence (AI) can date back to the era before the invention of modern computers, Embodied AI is usually vaguely defined with a relatively short history. It can refer to systems of different levels, ranging from any (semi-)autonomous systems involving physical structures with or without organism-like bodies [28]. The early studies are largely motivated by the fields of cognitive science and developmental psychology, where it is hypothesized that physical bodies and its complicated interactions with the surroundings promote the emergence of human intelligence, a general idea in embodied cognition [3]. Other studies [153] based on how babies learn suggest that the rich regularities inherently presented in the physical world help to shape intelligence, and such a process is accelerated by the agents' sensorimotor activities in exploring the surroundings.

Nowadays, Embodied AI is generally defined as the study of AI agents operating in 2D/3D physical environments that receive perceptual inputs and output actions to perform certain tasks. For its multidisciplinary nature, Embodied AI usually involves researchers from a wide range of fields such as computer vision, natural language processing, reinforcement learning, graphics, simulation, robotics, and so on. More concretely, Embodied AI is the study of intelligent agents that can see (usually in an egocentric view), talk (via languages or audios), reason (understand the surroundings and plan), and act (through motor controls or high-level actions). Some representative tasks include visual-and-language navigation [4], visual language task completion [149], object manipulations [107], and embodied question answering [37].

1.2 Problem Formulation in Embodied AI

Researchers in Embodied AI often adopt a unified formulation that captures the essence of most tasks in this field. Specifically, we introduce the partially observed Markov decision process (POMDP) [99] setup from the reinforcement learning (RL) literature. Namely, a POMDP is a representation of an environment as a 9-tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, p(s), T, \mathcal{R}, \rho_0, \gamma, H)$, where $\mathcal{S}$ is the state space, $\mathcal{O}$ the observation space (the actual perceptual signals received by the agent), $\mathcal{A}$ the action space, $p(s)$ the observation distribution conditioned on the state $s \in \mathcal{S}$, T the dynamic transition distribution, $\mathcal{R}$ the reward function that incorporates any goals required by the underlying tasks, ρ_0 the initial state distribution, and γ the discount factor. We usually assume a finite horizon H since in tasks in Embodied AI should be finished within a limited time.

For each episode or instance of the task $\mathcal{T}$ in Embodied AI, the agent is spawned and the environment is re-configured with state s_0 sampled from the distribution ρ_0. At each timestamp t, the agent receives an observation (can be visual, audio, etc.) $o_t \sim p(s_t)$. The partial observability setup is common due to the egocentric perception of the agent or the contextual information being revealed partially at each time. It then executes an action $a_t \sim \pi(s_t)$ according to some policy π (can be motor activities, text outputs, etc.), receives a reward $r_t \sim \mathcal{R}(a_t, s_t)$ which is determined by the underlying task $\mathcal{T}$, and reaches a new state $s_{t+1} \sim T(s_t, a_t)$. By such policy π we can obtain a trajectory $\tau = s_0, s_1 \sim T(s_0, a_0 \sim \pi), ..., s_{H+1} \sim T(s_H, a_H \sim \pi)$. The agent solves the task $\mathcal{T}$ by trying to learn the optimal policy π^* that maximizes the expected return (sum of discounted rewards over a trajectory), i.e.,

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} \sum_{t=0}^{H} \gamma^t r_t$$

In many cases, the metric for the task is given by a binary variable, indicating whether it succeeds or not. This is equivalent to using a binary threshold on the sum of the discounted reward.

1.3 Simulators

Almost all recent research in Embodied AI in building embodied agents involves the use of simulators. There are multiple obvious advantages of training agents in simulated environments that even make it arguably necessary. First of all, the sheer amount of physical resources (robots, indoor scenes, etc.) required for training is usually prohibitively expensive. Secondly, training in simulated environments significantly improves training speed as it bypasses physical constraints in the real world. Thirdly, in the real world, it is hard to reset the physical states of the environments such as the positions and poses of 3D objects exactly. There are various existing simulators with different characteristics. These diverse features play a major role in shaping the research in Embodied AI as different features are suited for different tasks. We will briefly discuss some of these key characteristics, including physics, rendering, speed, and controllers.

At a high level, a simulator has two major components: an engine that simulates the physical states of the environment (which is a dynamic model that generates s_{t+1} given s_t) and a renderer that provides multi-modal sensory signals that the agents actually observe (which is a model that generates o_t given s_t). Dynamics modeling of the physical state transitions is the most critical part of simulators as most Embodied AI tasks entail interactive environments. Due to the infeasibility of a perfect simulation of the real world, there exist various levels of accuracy and details for modeling real-world physics. Typically, the designers of the simulators make such decisions as different levels are suitable for different tasks in Embodied AI. The simplest level could support minimal physical simulation, for instance, a static maze that only supports tasks such as navigation. A basic level includes features such as collision modeling and rigid-body dynamics. More advanced ones are equipped with features such as enhanced modeling of articulated objects, soft-body physics on deformable objects, fluid dynamics, and differentiable physics (e.g., DiffTaichi [66]). The property derived from the different levels of physical simulation is the supported action space. Some simulators support fine-grained low-level motor controls while others only support high-level actions as they lack such simulation capabilities.

As for rendering, 2D images are the most common rendered outputs of the simulators. Different simulators achieve different levels of photorealism. Compared to the pioneer simulator DeepMind Lab [5] in Embodied AI research, most recent simulators can generate images of rather high resolution and fidelity. Advanced techniques such as ray tracing and physically-based rendering are also widely adopted. More recently, researchers have paid more attention to the fidelity of other modalities of sensory signals. For instance, SAPIEN incorporates advanced material-dependent noise-aware rendering for its depth signals [188]. ThreeDWorld [50] features a high-fidelity audio rendering of the environments.

The speed of a simulator is an important metric, which depends on both the physical simulation process and the rendering process. Solving most tasks in Embodied AI requires a large number of interactions ("samples" using words in RL). A faster speed enables extensive usage of policy gradient-based methods which typically entail enormous amounts of samples. On the other hand, a slower speed restricts the tools yet encourages the development of more sample-efficient and generalizable algorithms. However, it is rather challenging to give a precise comparison of the speed across different simulators, because there are a variety of factors to be considered that can influence the absolute speed, which is normally measured in frame rate, i.e., frame per second (FPS), for instance, the specific hardware setups (modern GPUs are much faster than previous ones) and number of threads or processes used. In terms of physics simulation, there are no standard scenes and object sets that are used to benchmark the speed (complicated scenes take much longer to simulate). Moreover, the speed-accuracy trade-off is inherent in every physics engine - choosing a smaller Δt means a slower but more accurate simulation. Usually, a simulation speed of around 1000 FPS allows large-scale RL, and faster ones indicate that simulation might not be the bottleneck for model training. In terms of rendering, there are no standard setups for resolution, image quality, and specific types of sensors.

There are multiple types of controller interfaces between the user and the simulator, which determine how easily the researchers can collect training data. The most common one is through API calls (usually via Python for programs and via keyboards/mice for humans). Some simulators have advanced controllers that support virtual reality (VR) interfaces, which significantly increase the efficiency and effectiveness of human control. Large-scale high-quality demonstrations for solving tasks in Embodied AI usually rely on VR-based teleoperation, which is both expensive and not widely supported in recent simulators. Another direction is to leverage vision-based teleoperation as the controller [168], which is still under-explored. Some simulators such as VirtualHome [122] support unconventional controllers including human language, which is easier to obtain yet the accuracy of the control highly depends on the quality of the command and effectiveness of the language parser.

1.4 Datasets and Assets

Datasets and assets are also critical components of Embodied AI research. Here we refer to the physical or virtual assets (such as robots and 3D objects) and the collected demonstrations. While some are built-in in their associated simulators, many of those are proposed to be simulator-agnostic: they are used for multiple different tasks and across different simulators.

Agents in simulation usually interact with the environments via a virtual yet physical robot. Different mechanical constraints of various robot actuators give us a wide range of robots. Some robots are highly realistic and are reflective of their real-world counterparts. For instance, AI2-Thor supports LoCoBot[1], Habitat 2.0 supports mobile manipulators like Fetch[2], fixed-base arms like Franka[3] and quadrupeds like AlienGo[4]; RLBench [71] is based on Franka; DoorGym [166] is based on Berkeley BLUE Robot arm [51]; simulators such as Meta-World [182] utilize MuJoCo for simulation and so are equipped with robot models such as Sawyer[5]. On the other hand, some robots do not resemble those in the real world: the humanoid robot in iGibson 2.0 [95], the flying gripper in SAPIEN, and the magnet-based robot Magnebot in ThreeDWorld. The different robot assets have a major impact on the action space of the tasks. For instance, some simulators primarily support high-level actions (discrete, magic grasp, etc.) which are good for learning long-horizon tasks but cannot be directly transferred to the real world where it requires gross motor control (low-level actions).

Object-oriented assets aim to promote research in object recognition, pose estimation, attribute learning, object manipulation, and other aspects related to Embodied AI research. The iTHOR assets in the AI2-Thor framework [87] consist of more than

[1]http://www.locobot.org
[2]https://fetchrobotics.com
[3]https://www.franka.de
[4]https://www.unitree.com
[5]https://www.rethinkrobotics.com/sawyer

100 interactable 3D objects that support rich physical properties. The YCB Benchmark [14] standardizes both the 3D models of the objects and their real-world counterparts (the physical copy) by scanning 3D everyday objects in the real world. More recently, PartNet-Mobility Dataset [174] which is used in various simulators is designed to enable more complex manipulation of articulated objects.

Scene-oriented assets typically feature diverse indoor scenes that are either static (supportive of navigation-based tasks) or compatible with interactable objects. In either case, they usually promote tasks of a longer horizon than solely object manipulations, such as mobile manipulation (i.e., navigation + manipulation) and object rearrangement. There are three types of scene assets: synthesized, reconstruction-based, and floor plan-based. They reflect the different trade-offs between scale, diversity, and quality of the generated scenes. The creation of synthesized scenes usually involves professional 3D artists and is very expensive (e.g., the construction of interactive ReplicaCAD requires $>$ 900 hours of human effort). The 3D reconstruction-based scenes are created by leveraging 3D scans of real buildings. Yet due to the imperfection of 3D reconstruction methods, they might contain artifacts and errors. Floor plan-based scenes are generated by first producing the floor plans and then converted to full 3D models (usually by some heuristics). It is relatively inexpensive to obtain (e.g., LUMINOUS [190] adopts procedural generation) with some compromise of the scene quality.

Language-oriented datasets refer to those where human language plays a central role. They inherently consist of multiple input modalities compared to the pure language-based datasets in the field of NLP. For instance, Cornell Instruction Following Framework (CIFF) [105] is an early effort to provide a unified framework for benchmarking instruction following tasks (consisting of mostly navigation and simple manipulation). ALFRED [149] is a more recent instruction following dataset for more complex navigation and object manipulations (e.g., cut an apple, cook an egg in the microwave). On the other hand, some datasets consider languages as actions to interact with the environments. For instance, the EmbodiedQA [171] is used to train an agent to navigate in the environment to answer a question about the scene. A later dataset, named Vision-and-Dialog Navigation (CVDN) [165], allows dialogues from the agents to clarify the tasks. However, the dialogues cannot be generated from the oracle in real-time and so methods designed for CVDN can only leverage pre-defined conversations. Due to the nature of language, most existing assets do not allow online conversation. Nevertheless, it becomes feasible to do so with the recent advances in large language models [113].

Skill-oriented datasets typically benchmark locomotion or object manipulation skills of robots with additional collected demonstrations so that AI agents can leverage a learning-from-demonstration setup. For instance, ManiSkill [107], RLBench-100 [71] and MT-50 [182] promote object manipulation research. The Advanced Physical Prediction Benchmark proposed in ThreeDWorld evaluates a set of more general skills (such as object permanence and collision understanding) of an agent.

1.5 Embodied AI Tasks and their Key Challenges

In this section, we briefly introduce the three major categories of tasks in Embodied AI, including navigation, object manipulation, and instruction following.

Navigation is a group of tasks where the agent is asked to navigate through (indoor) scenes to reach a specified target (a relative location coordinate, a semantic class, a short sentence specifying the location, etc.). While the ground truth map is not provided, the challenge is to understand the scenes and keep track of where the agent has visited and where it should explore next (similar to SLAM). The tasks usually also emphasize the efficiency of the navigation, measured by the ratio between the length of the actual trajectories and that of the shortest possible ones. It can be further divided into visual navigation and visual language navigation (VLN) where the latter also asks the agents to understand human language as part of the task specification. VLN can be extended to also allow dialogues between the agent and the environment (e.g., the instructor) to eliminate any ambiguities in the instructions.

Manipulation asks the agents to interact with objects in the scene to complete certain tasks. It can be with static robotic arms (static manipulation) or with mobile ones (mobile manipulation). Based on the type of objects, it involves rigid and articulated objects or deformable objects (soft-body manipulation). The specific atomic actions that constitute the tasks are usually push, pull, grasp, pour, etc. The major challenges lie in high precision perception and control (e.g., plug charger requires a precision level of around 1mm), the handling of geometric variations (e.g., common objects such as chairs have diverse shapes), and the difficulties of modeling certain objects' dynamics (e.g., cloth).

Instruction following is an umbrella term for mobile manipulation tasks with human instructions as the task specification. The agent is asked to translate a potentially long sequence of human instructions into actions (e.g., ALFRED). Besides the requirement of interpreting human instructions correctly, the two lines of work have their different challenges. Model-based approaches where the agent acquires a map of the surrounding [7, 104, 76] entail affordance-aware scene understanding. Purely instruction following ones where the agent directly translates the instructions [110, 115, 156] thrive given a tremendous amount of training data (which is usually not the case).

1.6 Introduction to the Presented Work

Decomposing a complex task into a set of simpler ones is a common technique used by AI agents, especially in the field of Embodied AI. Intuitively, humans routinely perform decompositions of tasks into goals, subgoals, and low-level actions, to improve the effectiveness or the efficiency [35]. In this dissertation, we present three methods, each from a different perspective, that adopt the general idea of task decomposition to handle some of the aforementioned challenges and improve robot learning.

The first work to be presented (in Chapter 2) is regarding acquiring embodied agents, solely from offline data, that follow human instructions to complete long-horizon household tasks (the ALFRED challenge). The nature of this task (long sequences of sub-tasks, each composed of high-level actions, though) makes it suitable for task decomposition. Its challenges include the requirement of an affordance-aware scene understanding of the indoor environment and the compositional generalizability in executing a sequence of diverse actions. To deal with this, we propose to decompose the tasks twice. Firstly, we decompose the task into exploration and execution phases. In the first phase, we utilize multimodal signals to explore the scene in a task-driven manner to obtain an affordance-aware semantic map that can be used for downstream task planning. In the second phase, we adopt a hierarchical task execution system to interact with the environments according to the semantic map and the instructions. We design the task execution system to operate in a sub-task manner by further categorizing the sub-tasks into either navigation or object manipulation. Our work achieves a substantial improvement in generalization over the unseen combinations of sub-tasks in unseen scenes.

The second work to be presented (in Chapter 3) handles short-horizon tasks. As pointed out by Moravec's paradox, low-level control tasks are usually harder for AI agents to learn than high-level reasoning tasks. We study challenging contact-rich object manipulation with high-precision and/or object-level geometric variations. We find that it is beneficial for short-horizon tasks to be further decomposed into subskills (e.g., atomic actions). Accordingly, we develop a novel hierarchical imitation learning method that can utilize scalable, albeit sub-optimal, demonstrations. We first propose an observation space-agnostic approach that efficiently discovers the multi-step subgoal decomposition (sequences of key observations) of the demos unsupervisedly. By grouping temporarily close and functionally similar actions into subskill-level segments, the discovered breakpoints (the segment boundaries) constitute a chain of planning steps (i.e., the chain-of-thought) to complete the task. Next, we propose a Transformer-based design that effectively learns to predict the decomposition (the chain-of-thoughts) as the high-level future guidance for low-level actions by using prompt tokens and a hybrid masking strategy during training. Our model consistently surpasses existing strong baselines.

Besides the aforementioned hierarchical approaches where tasks are decomposed temporarily into sub-tasks, task decomposition can also refer to partitioning the task space itself. While generalization in deep RL over unseen environments requires large-scale policy optimization over diverse environment variations, the optimization process is already computationally prohibitive. One of the key ideas of the third work to be presented (in Chapter 4) is to decompose the space of task variations and learn to solve the task in a divide-and-conquer manner. Specifically, we observe that an agent (a generalist) trained on many variations tends to learn faster at the beginning yet plateaus at a less optimal level for a long time. In contrast, an agent (a specialist) trained only on a few variations can often achieve high returns under a limited computational budget. We, therefore, propose an RL framework to tackle complex tasks in a distributed training manner to have the best worlds of both worlds. Several key ablations reveal when and how to launch the specialist agents as well as how to merge them back into a single generalist agent. Our proposal is a meta-framework that brings performance boost to various baseline online RL methods over several popular RL benchmarks.

1.7 Additional Work Done During my PhD

Apart from the aforementioned research, I have been working on diverse topics within AI/ML. Some of the work which I am the primary investigator includes: (1) the study of local minima in optimizing neural network where we propose a metric that is both strongly indicative of the generalizability of the neural network and may be effectively applied as a practical regularizer with both theoretical and empirical justifications [78]; (2) the study of robust image translation where we propose a novel multi-scale "semantic robustness" loss for GAN-based image translation models to reduce semantics flipping that is common in unpaired image-to-image translation tasks [79]; (3) the first empirical study of image advertisement understanding through the lens of large vision-language models, where we augment these models with real-world knowledge [77].

Chapter 2

Task Decomposition for High-level Long-horizon Instruction Following

Several challenges in solving long-horizon embodied multimodal tasks include long-horizon planning, vision-and-language grounding, and efficient exploration of the indoor scene. In this work, we identify a critical bottleneck, namely the performance of planning and navigation, and tackle it by decomposing the task into a task-driven multimodal exploration phase and a hierarchical planning and execution phase. Specifically, we propose a Neural SLAM approach that, for the first time, utilizes several modalities for exploration, predicts an affordance-aware semantic map, and plans over it simultaneously. This significantly improves exploration efficiency, leads to robust long-horizon planning, and enables effective vision-and-language grounding. With the proposed Affordance-aware Multimodal Neural SLAM (AMSLAM) approach, we obtain more than 40% improvement over prior published work on the ALFRED benchmark and set a new state-of-the-art generalization performance at a success rate of 23.48% on the test unseen scenes[1].

2.1 Introduction

There is significant recent progress in learning simulated embodied agents [115, 189, 8, 108, 152, 156] that follow human language instructions, process multi-sensory

[1]The code is publicly available at https://github.com/amazon-research/multimodal-neuralslam.

inputs and act to complete complex tasks [4, 37, 21, 149]. Despite this, challenges remain before agent performance approaches satisfactory levels, including long-horizon planning and reasoning [8], effective language grounding in visually rich environments, efficient exploration [24], and importantly, generalization to unseen environments. Most prior work [152, 115, 110, 156] adopted end-to-end deep learning models that map visual and language inputs into action sequences. Besides being difficult to interpret, these models show limited generalization, suffering from significant performance drop when tested on new tasks and scenes.

In contrast, hierarchical approaches [189, 8] achieve better generalization performance and interpretability. Although hierarchical structure is helpful for long-horizon planning, its key impact is an expressive semantic representation of the environment acquired via Neural SLAM-based approaches [17, 20, 8]. However, a missing component in these methods is fine-grained affordance [85, 124]. To build a robotic assistant that can follow human instructions to complete a task (e.g., *Open the fridge and grab me a soda*), it is essential that the agent can perform affordance-aware navigation: it must navigate to a reasonable position and pose near the fridge that enables follow-on actions *open* and *pick-up*. Operationally, the agent has to move to a location where the fridge is within reach without preventing the fridge door from being opened. Ideally, it should also position itself so that the soda is in its first-person viewing field to allow the follow-on *pick-up* action. This is challenging compared to pure navigation (where navigating to any location close to the fridge is acceptable). To achieve this, we propose *an affordance-aware semantic representation that leads to accurate planning for navigation setting up subsequent object interactions for success.*

Efficient exploration of the environment [133, 24] needs to be addressed to establish this semantic representation - it is unacceptable for a robot to wander around for an extended period of time to complete a single task in a real-world setting. To resolve this issue, we propose *the first task-driven multimodal exploration module that takes language* *instruction as guidance and keeps track of visited regions to explore the area of interest efficiently.* This lays a foundation for map construction, which is critical to long-horizon planning.

Here, we introduce Affordance-aware Multimodal Neural SLAM (AMSLAM), which implements two key insights to address the challenges of robust long-horizon planning, namely, efficient exploration and generalization: **1. Affordance-aware semantic representation** that estimates object information in terms of where the agent can interact with them to support sophisticated affordance-aware navigation, and **2. Task-driven multimodal exploration** that takes guidance from language instruction, visual input, and previously explored regions to improve the effectiveness and efficiency of exploration. AMSLAM is the first Neural SLAM-based approach for embodied AI tasks to utilize several modalities for effective exploration and an affordance-aware semantic representation for robust long-horizon planning. We conduct comprehensive empirical studies on the ALFRED benchmark [149] to demonstrate the key components of AMSLAM, setting a new state-of-the-art generalization performance at 23.48%, a >40% improvement over prior published state-of-the-art approaches.

2.2 Related Work

Recent progress in Embodied AI, spans both simulation environments [87, 95, 140, 50, 122] and sophisticated tasks [37, 4, 149]. Our work is most closely related to research in language-guided task completion, Neural SLAM, and exploration.

Language-Guided Task Completion

. ALFRED [149] is a benchmark that enables a learning agent to follow natural language descriptions to complete complex household tasks. The agent's goal is to learn mappings from natural language instructions to a sequence of actions for task completion in a simulated 3D environment. Various modeling approaches have been proposed falling into roughly two families of methods. The first focuses on learning large end-to-end models that directly translate instructions to low-level agent actions [152, 156, 115]. However, these agents typically suffer from poor generalization performance and are difficult to interpret. Recently, hierarchical approaches [189, 8] have attracted attention due to their better generalization and interpretability. We also adopt a hierarchical structure, focusing on affordance-aware navigation thereby achieving significantly better generalization than all existing approaches.

Neural SLAM and Affordance-aware Semantic Representation

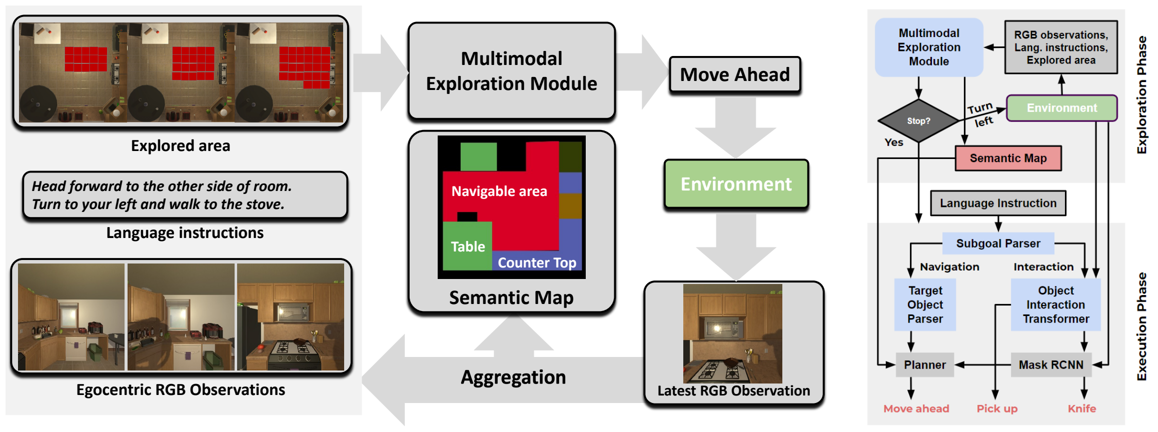
. Neural SLAM [17, 18, 20], constructs an environment semantic representation enabling map-based long-horizon planning [19]. However, these are tested in *pure* navigation tasks instead of complex household tasks, and does not consider affordance [124, 108, 175], which is *required* for tasks involving both navigation and manipulations. In [8], the authors utilize SLAM for 3D environment reconstruction in language-guided task completion. Their approach relies heavily on accurate depth prediction (less robust in unseen environments). Instead, we propose a *waypoint-oriented representation* which associates each object with the locations on the floor from where the agent can interact with the object. Furthermore, different from the 2D affordance map in [8] that directly predicts affordance type, our semantic representation supports more fine-grained control of the robot's position and pose, which facilitates significantly better generalization. The approach in [124] assumes direct access to the ground truth depth information (not available in our setup) and the method in [108] only focuses on pure navigation problems. Concurrently, in [104], the authors also propose constructing a semantic map and then planning over the map. However, their approach relies on depth prediction, which requires extra information from the environment during training.

Learning to Explore for Navigation

. An essential step in Neural SLAM-based approaches is learning to explore the environment for map building [133, 24, 74, 17]. Multiple approaches have been proposed to tackle aspects of exploration in the reinforcement learning [142, 117, 13, 24, 74], computer vision [133, 108], and robotics [8, 59] communities. The central principle of prior methods is learning to reduce environment uncertainty; different definitions of uncertainty lead to the following types of methods [133]. Curiosity-driven [142, 117, 13] approaches learn forward dynamics and reward visiting areas that are poorly predicted by the model. Count-based exploration [160, 6, 112, 134] encourages visiting states that are less frequently visited. Coverage-based [24, 74] approaches reward visiting all navigable areas by searching in a task-agnostic manner. In contrast, we propose a task-driven multimodal exploration approach utilizing egocentric visual input, language instructions, and memory of explored areas to reduce task-specific uncertainty of points of interest (areas important to complete the task). We show this to be more efficient, leading to more effective map prediction and robust planning.

2.3 Problem Formulation

We focus on the ALFRED challenge [149], where an agent is asked to follow human instructions to complete long-horizon household tasks in indoor scenes (simulated in AI2Thor [87]). Each task in ALFRED consists of several subgoals for either navigation (moving in the environment) or object interactions (interacting with at least one object). Language inputs contain a high-level task description and a sequence of low-level step-by-step instructions (each corresponding to a subgoal). The agent is a simulated robot with access to the states of the environment only through a front-view RGB camera with a relatively small field of view. The agent's own state is a 5-tuple (x, y, r, h, o), where x, y are its 2D position, r the horizontal rotation angle, h the vertical camera angles (also



Details of Exploration Phase Pipeline of AMSLAM

Figure 2.1. AMSLAM consists of two phases. **Exploration Phase**: The agent aims to explore the environment given guidance from low-level language instructions, egocentric observations, previous exploration actions, and the explored area. At the same time, it produces waypoint-oriented semantic maps. **Execution Phase**: Given the language instructions and the affordance-aware semantic representation (i.e., semantic maps) acquired during exploration, the agent executes the subgoals sequentially. It uses a planning module (which consumes the semantic map) for navigation subgoals and an object interaction transformer for other subgoals.

called "horizon") and o the type of object held in its hand. The state space of the agent is discrete, with navigation actions: `MoveAhead` (moving forward by $0.25m$), `RotateLeft` & `RotateRight` (rotating in the horizontal plane by 90°) and `LookUp` & `LookDown` (adjusting the horizon by 15°). Formally, $r \in \{0°, 90°, 180°, 270°\}$, $h \in \{60°, 45°, ..., -15°, -30°\}$ where positive h indicates facing downward. With these discrete actions, the agent has full knowledge of the relative changes $\Delta x, \Delta y, \Delta r$ and Δh. Each of the 7 object interaction actions (`PickUp`, `Open`, `Slice`, etc.) is parametrized by a binary mask for the target object, which is usually predicted with a pre-trained instance segmentation module. Featuring long-horizon tasks with a range of interactions, the ALFRED challenge evaluates an agent's ability to perform tasks over *unseen test scenes*, while only allowing ≤1000 steps and ≤10 action failures for each task at inference time.

2.4 Affordance-aware Multimodal Neural SLAM

Affordance-aware navigation is a major challenge in solving complex and long-horizon indoor tasks such as ALFRED with both navigation and object interactions. Specifically, given each object of interest in the scene, the agent is required to not only find and approach it but also end up at a pose (x, y, r, h), that is feasible for subsequent interactions with the object. For instance, to open a fridge, the robot should approach the fridge closely enough (so the door is within reach), look at it (so that the fridge is in the field of view), and leave enough room to open the door. To solve a long-horizon task involving multiple navigation and object interaction subgoals, it is natural to use an explicit semantic map, either 2D or 3D, of the environment (similar to Neural Active SLAM [17]), together with model-based planning (e.g. as in HLSM [8]). This line of work tends to generalize better than models that directly learn mappings from human instructions to navigation & interaction actions (e.g., E.T. [115]). With perfect knowledge of the environment, it is possible to achieve (nearly) perfect performance. In practice, however, the semantic map acquired at inference time is usually far from ideal, primarily due to **Incompleteness** (missing information due to insufficient exploration of the scene) and **Inaccuracy** (erroneous object location prediction on the map, especially for small objects).

To improve exploration performance, we propose a multimodal module that, at each step, predicts an exploration action $a \in \{\texttt{MoveAhead}, \texttt{RotateLeft}, \texttt{RotateRight}\}$ by taking visual observations & actions in the past, step-by-step language instructions, and the explored area map which indicates where the agent has visited. We show that, compared to existing model-based approaches on ALFRED (e.g., HLSM [8] which applies random exploration), our use of low-level language instructions leads to more efficient exploration. The proposed exploration module operates at the subgoal level and only predicts exploration actions (in contrast to E.T. which *directly predicts actions for the* *entire task*). The extra modality (the explored area) facilitates exploration by providing the agent with explicit spatial information. We illustrate the exploration module in Figure 2.3, elaborate its details in Section 2.4.3, and empirically demonstrate its advantages in Section 2.5.

To deal with the inaccuracy in map prediction, we carefully design an affordance-aware semantic representation for the environments. On one hand, knowing the precise spatial coordinates of objects requires precise depth information, which is difficult to acquire due to 3D sensor noise and/or inaccuracy in predicting depth from 2D images. On the other hand, affordance-aware navigation essentially asks for poses (x, y, r, h) of the agent suitable for interactions with the target objects, thus requiring only coarse-grained spatial information. Given an object type o, we define such corresponding poses as *waypoints* $\mathcal{W}_o$ and then treat navigation as a path planning problem among different waypoints. To generate such waypoints, we handle large objects (fridges, cabinets, etc.) and small objects (apples, mugs, etc.) differently. The waypoints for large objects are computed using 2D grid maps predicted and aggregated from front-view camera images by a CNN-based network; for small objects, we directly search over all observations acquired during the exploration phase with the help of a pre-trained Mask RCNN [60] (detailed below in Section 2.4.2).

2.4.1 Overall Pipeline

We illustrate the overall inference pipeline of our proposed framework in Figure 2.1. Given a task $\mathcal{T}$ specified by a high-level goal description and low-level human instructions, our method proceeds in two phases: exploration and execution. During exploration, the agent navigates across the room (guided by the language instructions) for a sufficient exploration of the indoor scene, where a multimodal transformer predicts the exploration actions sequentially. In the meantime, an affordance-aware semantic representation is acquired given the egocentric observations (images) at each step by a neural SLAM system.

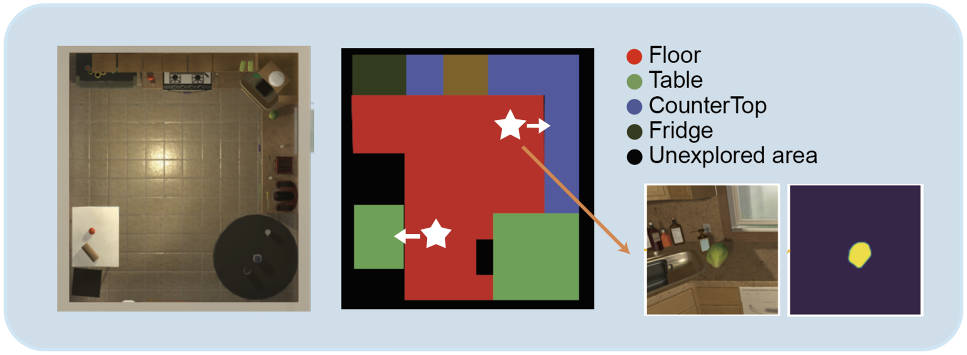


Figure 2.2. Illustration of the affordance-aware semantic representation used in our framework. **(left)** A top-down view of an indoor scene used in ALFRED (this view is not available to the agent at test time). **(middle)** A visualization of the corresponding semantic map; only shown are the side table (in green), countertop (in blue) and the navigable area (in red). Two waypoints (drawn in white stars and arrows) are displayed for the side table ($\in$ large object) and lettuce ($\in$ small object), respectively. **(right)** While the waypoint for the side table is computed from the predicted map, the waypoint for lettuce is obtained by searching among all exploration steps. The visual observation and its mask prediction on the waypoint for lettuce are shown. For reference, the high-level goal description of the task used in this example is "pick up the lettuce and place it on a table".

This representation can be used to derive *waypoints* for all target objects (from which the agent can interact with the objects of interest). When the exploration phase ends (by the agent predicting `Stop`), it moves to the execution phase, where the agent carries out actions predicted for each subgoal of the task $\mathcal{T}$ in a sequential manner. Specifically, since each step-by-step language instruction corresponds to a subgoal, we use a Transformer-based [167] subgoal parser to process the text and predict the subgoal $g \in \mathcal{T}$ is for navigation or not. For a navigation subgoal, we further use another Transformer-based target object parser to process the same text and predict the categories of the target objects, which are consumed together with the affordance-aware semantic representation by a Dijkstra-based planner to generate navigation actions. Otherwise, an object interaction transformer takes charge of the action predictions given the visual and language inputs of the object interaction subgoal.

2.4.2 Affordance-aware Semantic Representation

We empirically show in Section 2.5.1 that a major bottleneck for solving long-horizon navigation & interaction tasks is affordance-aware navigation. To do so, the agent needs a position and pose (defined as *waypoints* previously in this section) from which the potential follow-on actions for the target object are feasible, rather than the exact location of the target object. Accordingly, our goal is to develop a map representation that supports waypoint generation so that navigation can be solved reasonably well by path planning. ALFRED supports more than 100 types of objects (one way in which it mimics real-world complexity), and we propose to handle small and large objects differently. We detail our design below and give an example in Figure 2.2.

For a class c_{large} out of N_{large} large object types, we compute its waypoint $(x^{*}, y^{*}, r^{*}, h^{*})$ in 3 steps. **First**, we find all positions $\{(x, y)\}$ that might contain an instance of class c_{large} and all navigable locations for the robot, both represented on a 2D grid map of dimension $G \times G \times (N_{large} + 1)$, with grid size $G = 37$ and unit length $0.25m$. Each grid point in the map has a binary multi-hot vector to represent whether each object class appears there and whether the point is navigable. Specifically, at each (x, y, r) visited in the exploration phase, we use a pre-trained CNN (whose inputs are images at 3 different horizons observed at (x, y, r)) to predict a small partial map of the 2D map. We then aggregate across all exploration steps by max-pooling over these partial maps, each translated and rotated via a Spatial Transformer [69]. Similar to Neural Active SLAM [17], as we know the changes $(\Delta x, \Delta y, \Delta r)$ of the agent after each action, we can directly compute the parameters of the Spatial Transformer. After applying some post-processing, we obtain the final 2D map estimation. **Second**, we find (p_x, p_y) on this 2D map as the most confident position predicted for class c and then find the navigable position (x', y') closest to (p_x, p_y). **Third**, we choose the rotation r^{*} to be the one, suppose the agent stands at (x', y'), and an object at (p_x, p_y) appears closest to the center of the agent's field of view. To leave room for object interactions (by the agent's backing up a few steps), we compute $x^* = x' - \delta_1(c, r^*)$ and $y^* = y' - \delta_2(c, r^*)$ where δ_* are rule-based functions. We defer the estimation of the horizon h^* (the camera angle relative to the horizontal) to the execution phase described in Section 2.4.4.

For a small object type c_{small}, predicting its 3D coordinates precisely is rather challenging (even 2D object detection for small objects is hard [86]). In ALFRED, this is especially true since only RGB images are given at test time and many types of small objects occur rarely. To deal with this challenge, we propose to directly find the waypoint (x^*, y^*, r^*, h^*) for c_{small} by searching through all observations (RGB images) at each step during exploration. Specifically, we compare all instance masks for c_{small} predicted by a pre-trained Mask RCNN [60] that are of confidence $\geq \tau_c$. Then (x^*, y^*, r^*, h) is computed as the one where the aforementioned mask prediction has the largest area. Similar to the waypoint generation for large objects, we do not estimate the horizon h^* until the execution phase. Directly finding the waypoint of c_{small} (without estimating its location) relies heavily on how well the exploration is carried out (discussed later in Section 2.4.3). In case the agent finds no such waypoint (no valid observation of c_{small} during exploration), we instead use the waypoint for the container (normally of large object type such as fridge, side table) of the small object, which is much easier to find.

2.4.3 Task-driven Multimodal Exploration

The task-driven multimodal exploration module consists of several sub-modules, either learned or pre-trained/fixed. At a high level, given a task which is usually a sequence combining navigation and object manipulation subgoals, in the exploration phase, the agent goes through its navigation subgoals one by one to explore the scene. Note that we adopt a subgoal parser to predict whether a subgoal is for navigation or not based on the input instruction. Specifically, the module predicts exploration actions $a \in \{\texttt{MoveAhead}, \texttt{RotateLeft}, \texttt{RotateRight}\}$ or `Stop` auto-regressively for each navigation

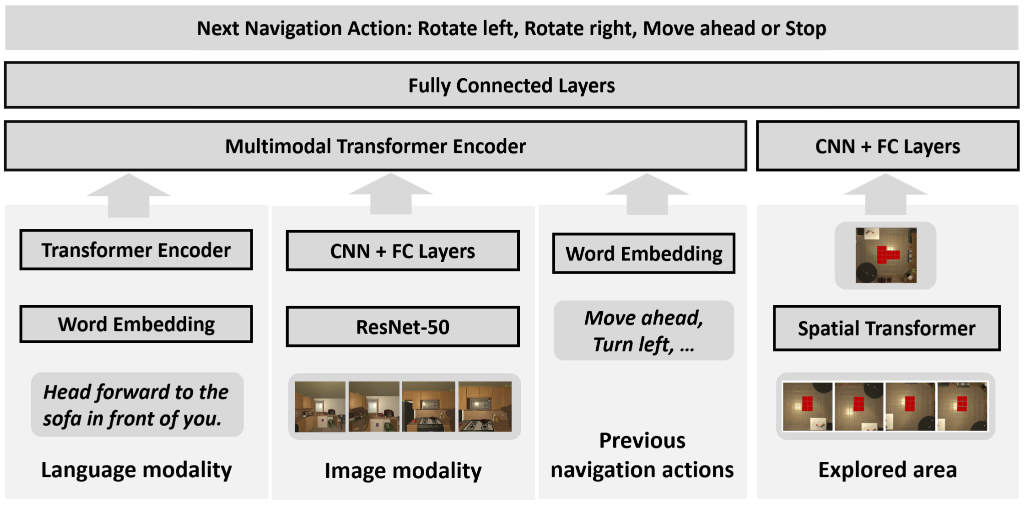


Figure 2.3. Consisting of several transformer-based networks, the exploration module operates at a subgoal level. Given a navigation subgoal during inference, it takes multiple modalities as input including language instructions for the current and subsequent subgoals, egocentric observations from the agent, and exploration actions in the past as well as their corresponding explored area. It predicts the next exploration action carried out in the environment auto-regressively. Illustrated is an example where given 4 previous actions the model tries to predict the 5-th. Note that for brevity the positional (and temporal) encoding layers right before the transformer encoders are omitted in the figure.

subgoal. The agent switches to the next navigation subgoal whenever it predicts `Stop` until the last one to end the exploration phase. Instead of random exploration (as in HLSM) or exploration for maximum coverage (as in Active Neural SLAM), our module utilizes low-level language instructions to achieve task-driven exploration for better efficiency. In ALFRED the exploration is done individually for each task and the steps count towards the total steps (which has a limit of 1000).

There are 4 modalities (and 4 corresponding branches). At each time step, the first branch takes two low-level human instructions, one for the current navigation subgoal, and one for the subsequent object interaction subgoal. The second and third branches consume the egocentric observations (images) and the previous exploration actions, respectively. Inspired by E.T. [115], we use a cross-modal transformer (also see [100, 189]), which aggregates inputs of these 3 modalities across all previous exploration steps to output

$f_{1,2,3}$. The fourth modality, the explored area, is a $G \times G$ 2D grid map marked 1 for the regions observed by the agent in the past and elsewhere 0, except that the center of the map (which always indicates the agent's position) is marked 2. This map is constructed in 2 steps. First, at each previous exploration step, since the agent can only observe a small area in front of it, we define a single-step explored region as a binary map where there is a 5×3 rectangle grid of 1's representing where the agent has observed. Second, we aggregate the maps across all previous steps. Each single-step map is translated and rotated by a Spatial Transformer and then merged by max-pooling (similar to the semantic map described in Section 2.4.2). Given this explored area map, we use a CNN to extract $f_4 \in \mathbb{R}^{32}$. Together with $f_{1,2,3}$, we finally use an MLP to predict the next exploration action. Illustrated in Figure 2.3 and evaluated in Section 2.5.5, our design to represent the action history explicitly and geometrically improves both the effectiveness and efficiency of the exploration.

We train the exploration module supervisedly. A common practice (e.g. in HLSM [8], EmBERT [156], LWIT [110]) during inference is to augment the exploration by injecting actions periodically. We manually inject 4 `RotateRight` after every 2 `MoveAhead` predicted by our module to acquire 360° views of the scenes. Moreover, the semantic representation produced by the neural SLAM requires input images of 3 different horizons. So we further inject two `LookUp` or two `LookDown` actions alternately after every exploration action. This zigzagging scheme is an efficient way to acquire images of multiple horizons, which only triples the total number of steps in the exploration phase. Since each exploration step counts towards the total steps for a task, there is a trade-off between exploration (a better view of the environment) and exploitation (there is an upper limit of 1000 for total steps).

2.4.4 Other Modules

Subgoal Parser & Target Object Parser

Both the subgoal parser and the target object parser take the language instruction as the only input. Both models use the same Transformer-based architecture (not sharing weights, though) and are trained supervisedly. The subgoal parser performs a binary classification, predicting whether each subgoal is for navigation (based on its human instruction). The target object parser predicts both the target object and its container (if it has one) by taking the language instructions for the navigation subgoal *and for the next subgoal*.

Online Planner for Affordance-aware Navigation

Given the affordance-aware semantic representation that supports waypoint generation, we deal with a navigation subgoal in 3 steps: (1) Obtain a waypoint $(x, y, r, *)$ for the object type predicted by the target object parser. (2) Derive an action sequence from the path connecting the current location and pose (x', y', r', h') to (x, y, r, h') using Dijkstra's algorithm. (3) Decide the horizon h by online exploration by first navigating to (x, y, r, h'). The agent then goes through 6 horizons $\{60°, 45°, ..., 0°, -15°\}$ (essentially a search over most horizons allowed for the agent) and obtains the mask prediction with confidence > 0.8 (selected from $\{0.5, 0.6, 0.7, 0.8, 0.9\}$ using the valid unseen data) of the target object type by a pre-trained Mask RCNN. Finally, we select h with the largest mask area. See ablation studies of this scheme in Section 2.5.5.

Object Interaction Transformer

We adopt the cross-modal transformer again at the subgoal level for object interaction subgoals, which maps language instructions and visual observations (only for the current object interaction subgoal) to actions. The module is trained by imitation learning on the ALFRED training data at the subgoal level.

Table 2.1. Ablation study results of the generalization performance of our framework with ground truth navigation (denoted GT navi.) inserted together with different types of pose perturbations. For reference, we also include results from HLSM [8], the previous state-of-the-art method on ALFRED. Note that the object interaction transformer is abbreviated as Obj. Int. Xformer.

	Valid Unseen (%)	
	Success Rate	Goal Cond.
HLSM [8]	11.8	24.7
GT navi. + Obj. Int. Xformer	64.6	74.2
GT navi. + Obj. Int. Xformer + rand. horizon	21.7	35.9
GT navi. + Obj. Int. Xformer + rand. displacement	47.3	65.4

2.5 Experiments

We evaluate our method for language-guided task completion on the ALFRED challenge [149], which supports task evaluation in unseen environments (i.e., room layouts), the main focus of our method. ALFRED provides both a validation (with ground truth provided) and a test set (evaluation occurs in the server) for tasks sampled in indoor scenes unseen during training. The training split contains 21,023 tasks sampled from 108 scenes (i.e., rooms), the valid unseen split and the test unseen split contain 821 tasks sampled from 4 scenes and 1,529 tasks sampled from 8 scenes, respectively. For all experiments, we report the commonly used evaluation metrics: the task level *Success Rate* and the subgoal level *Goal Condition*. Notice that each task in ALFRED allows up to 1000 total steps (including both exploration and execution phases in our method).

We organize the presentation of the numerical results into 3 parts. Firstly, we demonstrate that affordance-aware navigation is the major bottleneck for language-guided task completion. Secondly, we present our main numerical results showing that with better affordance-aware navigation, our method significantly outperforms previous state-of-the-art methods on ALFRED. Finally, we perform a series of ablation studies to justify our design.

2.5.1 Validation of the Need for Affordance-aware Navigation

We first validate the need for better affordance-aware navigation. To do so, we analyze the generalization performance of our approach by using ground truth actions for navigation subgoals instead (we still use actions generated by the object interaction transformer otherwise). Similar to solving indoor tasks in real-world settings, the major bottleneck in tackling ALFRED is affordance-aware navigation. A navigation subgoal succeeds only if the agent stays close enough to the target object (so that it is within reach), sets a camera angle so that the object is in the field of view, and leaves room for object articulation (e.g., open a door of a fridge). We perform several experiments on the valid unseen data of ALFRED with numerical results reported in Table 2.1. Specifically, when the ground truth navigation actions (denoted GT navi.) are used during inference, our framework achieves an extremely high success rate (64.6%). However, the performance drops significantly if perturbations (random displacement is adding ± 1 to the coordinates) are added to the target (x, y, r, h) of each navigation subgoal, verifying our claim about the need for affordance-aware navigation.

2.5.2 Validation of the Need for Hierarchical Task Execution

We further validate the need for the hierarchical approach (i.e., subgoal level task execution) adopted by our method where each subgoal is executed sequentially by either a Dijkstra-based planner (for navigation subgoals) or otherwise the object interaction transformer. Here we justify the use of the object interaction transformer, which is a cross-modal transformer trained at the subgoal level. Specifically, we pre-process the training data from the original training fold of ALFRED such that each trajectory contains inputs (low-level language instructions and visual observations) and ground truth actions for a single object interaction subgoal. We evaluate our proposed module on the valid unseen split of ALFRED. Compared to E.T.+, which adopts a cross-modal transformer trained in full task level and with extra synthesized data, our object interaction transformer generalizes better on nearly all subgoals, as reported in Table 2.2.

Table 2.2. Success rates (%) of all 7 object interaction subgoals evaluated on valid unseen data in ALFRED. Overall, our method (object interaction transformer) generalizes better than E.T.+.

	Valid Unseen (%)						
	Toggle	Pickup	Cool	Put	Heat	Clean	Slice
E.T.+ [115]	83.2	69.0	99.1	69.6	**99.3**	91.2	65.8
Obj. Int. Transformer (ours)	**86.1**	**72.0**	**100.0**	**75.3**	98.5	**91.7**	**72.1**

2.5.3 Main Results

We present the main results of our proposed method, evaluated on the test unseen and valid unseen data of ALFRED. Our framework uses no ground truth or metadata about the environment during inference. As introduced in Section 2.4.1, our framework performs exploration to acquire knowledge about the indoor scenes and then predict actions in a hierarchical approach with both a rule-based planner and a few learning-based modules. We set a new state-of-the-art performance with a substantial improvement (>40%) over previously published methods (see Table 2.3).

While AI2THOR adopts a discrete action/state space for the agent, we believe the idea behind our task-driven exploration design and waypoint-based semantic representation is applicable to generic embodied tasks. In the presence of motion noise and pose sensor noise, Active Neural SLAM shows that such inaccuracy can be solved to an acceptable level by simply learning a pose estimator.

2.5.4 Qualitative Evaluations

We also illustrate qualitative results from our proposed method compared to the E.T. baseline. We display a pair of trajectories predicted from both models (for ours, we

Table 2.3. Performance on valid & test unseen data from ALFRED. Our method achieves new state-of-the-art results with a substantial improvement over previously published methods.

	Test Unseen (%)		Valid Unseen (%)	
	Success rate	Goal Cond.	Success rate	Goal Cond.
EmBERT [156]	7.52	16.33	5.73	15.91
E.T.+ [115]	8.57	18.56	7.32	20.87
LWIT [110]	9.42	20.91	9.70	23.10
HiTUT [189]	13.87	20.31	12.44	23.71
ABP [84]	15.43	24.76	-	-
HLSM [8]	16.29	27.24	11.80	24.70
AMSLAM (ours)	**23.48**	**34.64**	**17.68**	**33.96**

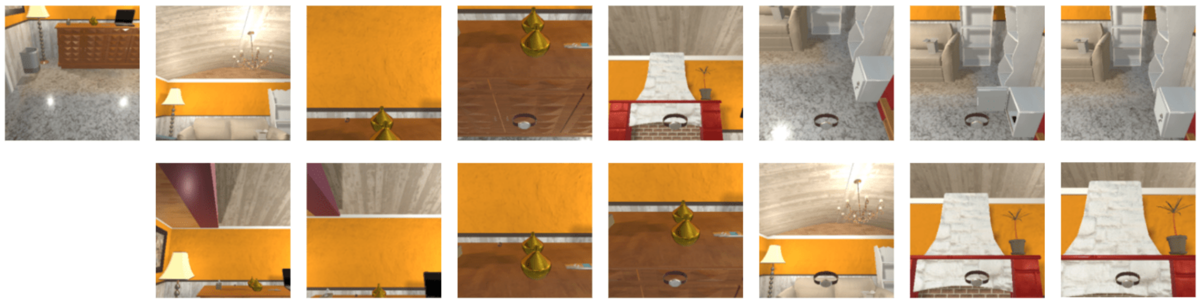


Figure 2.4. For the task "Move a watch to the inside of a small safe", the first row and second row correspond to trajectories predicted by ours and E.T. Each column corresponds to a different time step with $t = 0, 6, 12, 18, 24, 32, 35, 36$ from left to right.

only show results from the execution phase) for data in the valid unseen and test unseen split, respectively. By utilizing task-driven exploration to acquire the affordance-aware map of the scene, together with the planning and object interaction modules, our model is capable of completing long-horizon instruction-following tasks. In the first task where it asks the agent to grab a watch and then find and store it inside a safe, E.T. fails to navigate to a position such that it can successfully open the safe, whereas ours succeeds. In the second task for moving two spray bottles from a single shelf to the same toilet tank, E.T. fails to find the second sprayer while ours can move both sprayers to the right place. The two sets of trajectories are illustrated in Fig. 2.4 and 2.5, respectively.

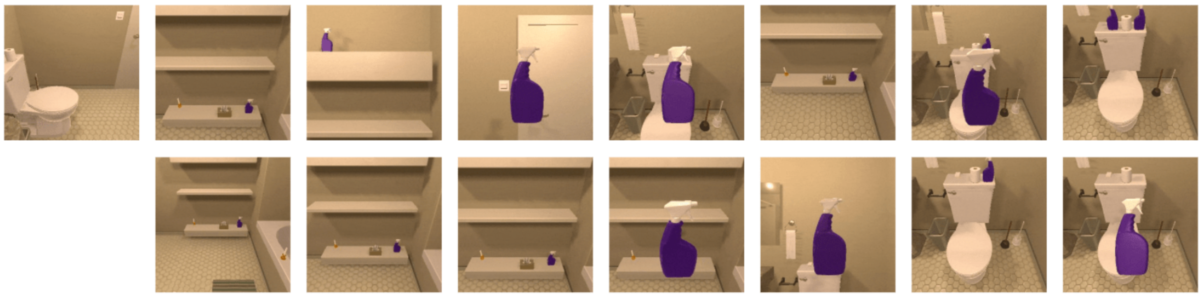


Figure 2.5. For the task "Place two spray bottles on a toilet tank.", the first row and second row correspond to trajectories predicted by ours and E.T. Each column corresponds to a different time step with $t = 0, 5, 10, 15, 20, 30, 35, 41$ from left to right.

2.5.5 Other Ablation Studies

We perform ablation studies to justify our framework design. In Ablation Studies I, we examine the components in our exploration module. We choose four variants. In variant AMSLAM + rand. exploration, we replace the task-driven multimodal exploration with a random exploration strategy similar to that in HLSM. In variant AMSLAM - lang., we do not use the language instructions at all to guide the exploration process In variant AMSLAM - lang. (partial), we only use language instructions associated with the navigation subgoal (i.e., without the next object interaction subgoal) to guide the exploration. In variant AMSLAM - explored area, we do not use the extra modality in the exploration module. For all variants, the exploration phase ends when a pre-defined upper limit of action fails or a pre-defined upper limit of exploration steps is reached. In Table 2.4, we compare the valid unseen performance on ALFRED, the coverage, defined as the number of distinct (x, y) pairs visited during the exploration phase (average per task), and the coverage efficiency (Cov. Eff.), defined as the coverage divided by total number of (un-augmented) exploration steps.

Specifically, for all the four variants, we stop the exploration process if (1) the module predicts a `Stop` action, or (2) the number of action failures reaches 4, or (3) the total number of exploration steps (including the injected actions) reaches 500. In the variant AMSLAM + rand. exploration, we adopt the following random exploration strategy. First, we make 4 `RotateRight` and adjust the horizon of the agent to acquire a 360° and 3 horizon view of the environment. Next, we obtain the navigable area estimated by our Neural SLAM system. Then, we randomly sample a point on the boundary of the navigable area and navigate to that point (during which process we inject the `RotateRight` and `LookUp/Down` similar to our multimodal exploration strategy). We repeat the previous step until the aforementioned stop condition is satisfied.

In Ablation Studies II, we examine the components in the affordance-aware map representation and in the online planner. We justify our approach of handling small (denoted as instance mask waypoints) and large objects (denoted as object map waypoints) in different ways by showing that the generalization performance degrades drastically when adopting only one strategy for waypoint generation. We show that the "backing up" rule in the planner (to leave room for articulated objects) is important. Moreover, we evaluate a variant (AMSLAM + rand. horizon) where horizon h is perturbed for all subgoals to justify our online exploration (and backtracking) strategy for finding the best h. Specifically, for the variant AMSLAM + rand. horizon, we first disable the online exploration and backtracking strategy of the planner for finding the best horizon h. We then randomly choose a final horizon for each navigation subgoal as one of $\{60°, 45°, ..., -15°, -30°\}$. See numerical results in Table 2.5.

Table 2.4. Ablation Studies I: the task-driven multimodal exploration module in AMSLAM.

	Valid Unseen (%)		Coverage Analysis (%)	
	Success Rate	Goal Cond.	Coverage	Cov. Eff.
AMSLAM + rand. exp.	9.73	22.10	20.40	58.39
AMSLAM - lang.	4.30	15.90	9.10	43.50
AMSLAM - lang. (partial)	13.66	28.93	24.37	66.05
AMSLAM - explored area	15.17	30.90	27.13	65.48
AMSLAM (ours)	**17.68**	**33.96**	**28.70**	**67.09**

Table 2.5. Ablation Studies II: affordance-aware semantic representation & planner in AMSLAM.

	Valid Unseen (%)	
	Success Rate	Goal Cond.
AMSLAM - object map waypoints	12.10	26.73
AMSLAM - instance mask waypoints	8.90	18.70
AMSLAM - back up steps	12.50	28.85
AMSLAM + rand. horizon	9.81	20.60
AMSLAM (ours)	**17.68**	**33.96**

2.6 Implementation Details

2.6.1 Neural SLAM module

CNN Architecture

The CNN used for predicting the 2D grid map of size $G \times G \times (N_{large} + 1)$ (with grid size $G = 37$ and unit length $0.25m$) has the following architecture. The inputs are 3 images, which are first processed by the ResNet-50 Faster RCNN feature extractor (we use the checkpoint provided by E.T.) Then the three 512-d features are each processed by the same 4-layer CNN (filter size 3, stride size 2, number of features as $256, 256, 256, 256$). The flattened and concatenated features from the previous step are then fed into a 4-layer FC network (number of features is $512, 512, 512, 128 \cdot 7 \cdot 10$). Next, the output features are reshaped into a $128 \times 7 \times 10$ tensor. Finally, another 3-layer CNN (filter size 3, stride size 1, number of features as $256, 256, N_{large} + 1$) takes in the tensor and outputs the $7 \times 10 \times (N_{large} + 1)$ map, which represents the prediction of a small region in front of the agent (in its egocentric view).

Aggregation

The 2D grid map predictions at each exploration step are aggregated via the use of a spatial transformer. The parameters to the spatial transformer are computed in the same way as the Neural Action SLAM (since we know exactly what each exploration action is and thus where the agent was headed). Each predicted 2D grip map is first rotated and translated by the spatial transformer (with the initial location considered as the center of the 2D map and the initial rotation angle considered as the direction facing upward) and then max-pooled to form the complete semantic map of the environment. During this process, since each region at a single step has a limited field of view, we mask the prediction at each step by a hard-coded 7×10 binary map (starting from the row the agent is standing at towards another 9 rows facing forward). We have tried multiple combinations for the shape of the binary map, with height $\in \{5, 6, 7\}$ and width $\in \{9, 10, 11\}$. We choose 7×10 using valid unseen data in ALFRED by the average prediction accuracy.

Model Training

We train the model by minimizing the cross-entropy distance between the ground truth semantic map and the predicted one. The training is performed for each single-step map prediction. The ground truth for the navigable area is generated by using the API from AI2Thor. The ground truth for the object map is not available for scenes in ALFRED, which uses a version of AI2Thor that does not support bounding box information for general objects in the scenes. However, later AI2Thor versions support such functionality. There are some scene layout mismatches between later versions of AI2Thor and the version used by ALFRED, though. We solve this by manually inspecting all 108 training scenes and fixing bugs by hand. We will release the code as well as the processed training data for our Neural SLAM module. We use the Adam optimizer to train the CNN model with an initial learning rate of 0.005 and a linear decaying schedule (starting from the second half of the training) to 0 for a total of 10 epochs. We find the best checkpoint using the prediction accuracy evaluated on the valid seen and valid unseen data of ALFRED.

Post-processing

To have a more robust navigable area for long-horizon planning, we further apply some post-processing steps to the aggregated navigable map by the Neural SLAM module (i.e., the last dimension of the 2D grid map). Specifically, we consider map A as the binary navigable area map where only the predicted confidence greater than 0.95 will be considered a valid prediction for a navigable point (i.e., a value of 1). We also consider map B as the binary map where a location with greater or equal to 3 nearby points (i.e., the one whose L1 distance to it is 1) being navigable (here we use confidence threshold 0.5) is marked as a navigable point. We then perform an element-wise product of the two maps to obtain the final navigable area.

2.6.2 Waypoint Generation

Small and Large Objects

As mentioned in the paper, we handle small and large objects differently when designing our affordance-aware semantic representation. In specific, we consider large object types in the following list and otherwise small object types:

- armchair, chair, cart, sofa, shelf, drawer, cabinet, countertop, sink, stove burner
- fridge, bed, dresser, toilet, bathtub, ottoman, diningtable, sidetable, coffeetable, desk

The Backing up Steps

The benefit of handling large objects in a 2 step process is that we can compute their waypoints in a more flexible manner. In our framework, we find backing up a few steps to be a very effective strategy, which leaves some margin between the target position of the agent and the target object the agent needs to interact with This strategy is particularly critical for articulated objects. We use simple heuristics, denoted δ_1 and δ_2 as introduced in the paper. Specifically, the agent will back up 3 steps if the target object is a fridge, 2 steps if it is a safe, a cabinet or a drawer, and 1 step for everything else. The implementation of the $\delta_*(\cdot, \cdot)$ is simply an integer $1, 2$ or 3 for either x or y coordinate given the 4 different rotation angles $r \in \{0°, 90°, 180°, 270°\}$.

2.6.3 Task-driven Multimodal Exploration Module

The Extra Modality: Explored Area

The extra modality introduced in our multimodal exploration module essentially tracks the action history explicitly and geometrically. Specifically, during each step in the exploration phase, we hard-code a 5×3 binary mask to indicate where the agent has observed in the current egocentric view. We find the exact shape of this region does not matter much (we have tried 3x2, 4x2, 5x4) as long as it helps to track where the agent has visited. As the agent always stands at the center of the explored region map, we set the binary mask as starting from the center row and extending facing forward towards another 4 rows. Then we merge these single-step explored areas by using the spatial transformer and max-pooling, the same way as when we aggregate the 2D object map in our semantic representation. We finally mark the center of the aggregated explored area as 2, indicating where the agent is standing. An illustration is shown in Figure 2.6 (for simplicity, we only draw a 3×2 binary mask), where four single-step explored area maps are aggregated into one. The CNN used to process the explored area is of 4 layers (filter size 3, stride size 2, number of features as $128, 128, 64, 32$). The output of the CNN is flattened and fed into a 2-layer FC network (the feature dimensions are 128 and 32). Then we concatenate its output with the output from the multimodal transformer (which extracts features for the other 3 modalities) and feed it into the final 3-layer FC network to predict the next exploration action (number of features are $256, 128, N_{exp} + 1$ where $N_{exp} = 3$ and the extra 1 is for the `Stop` action).

Training Data

We regenerate the trajectories from the training data of ALFRED by ignoring all object interaction actions and `LookUp/Down`. We train the exploration module by imitation learning on this new training set. Each sample in the new training set corresponds to one navigation subgoal in a trajectory of the original training set. As each trajectory in the

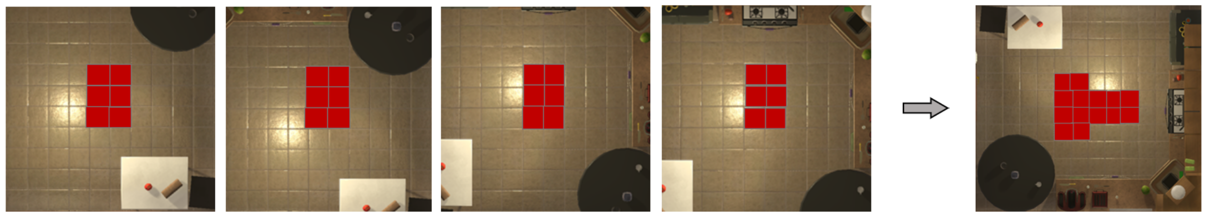


Figure 2.6. An illustration of the single-step explored area for four exploration steps (**left**) and the aggregated one (**right**). The actual size of a single-step explored area is 5×3 instead of 3×2 shown here.

training data in ALFRED starts with an initial horizon of 30°, all visual observations used for training are of such horizon (i.e., vertical camera angle).

Hyper-parameters

We train the exploration module by Adam optimizer with an initial learning rate of 0.001 and a linear decaying schedule (kicking in only for the second half of the training) to 0 for a total of 20 epochs. We find the best checkpoint using the coverage and coverage efficiency computed on the valid seen and valid unseen data of ALFRED.

2.6.4 Action Augmentation during Exploration

Zigzagging

At inference time, we inject two `LookUp` or two `LookDown` actions alternately after every exploration action so that we acquire images of 3 different horizons at each (x, y, r) observed during exploration. We choose this zigzagging scheme as it is the most efficient way to perform exploration in the vertical camera angle space. An example is listed below with the action sequence (after the periodic injection of `RotateRight`) shown in the first line and the zigzagged sequence shown in the second line.

- Move, Right, Move, Right, Right, Right, Right, Move, Left
- **Down, Up, Up**, Move, **Down, Down**, Right, **Up, Up**, Move, **Down, Down**, Right, **Up, Up**, Right, **Down, Down**, Right, **Up, Up**, Right, **Down, Down**, Move, **Up, Up**, Left, **Down, Down**

The injected actions are bolded, with the first `LookDown` inserted to handle the beginning of the exploration.

Other Details

Since each augmented trajectory (i.e., the sequence injected with `RotateRight` and `LookUp/Down`) is a strict superset of the original unaugmented ones predicted directly via the multimodal exploration module, these injection does not interfere with the normal inference pipeline of the exploration module. Specifically, we mask out the inputs (observations, actions history, and the explored area) corresponding to the injected actions in all of the 4 branches.

2.6.5 Subgoal Parser and Target Object Parser

Network Architecture

We use a transformer-based architecture similar to the multimodal transformer used for exploration. Since the input to both the subgoal parser and the target object parser is the language instruction, we mask out the inputs to the other 3 branches (i.e., we use uni-modal transformers). The two models share the same architecture while not sharing the weights. The subgoal parser is essentially a binary classifier. The target object parser, on the other hand, predicts two pieces of information, namely the target object for the navigation subgoal and its container (if it has one). This prediction involves two steps; we, therefore, model it in an auto-regressive manner.

Model Training

We train the subgoal parser and the target object parser by minimizing the cross-entropy loss using the ground truth object information. Specifically, we use APIs provided by AI2Thor to acquire spatial relationships and use them to decide the container object type for each instance of the small objects (which the target object parser is trained to predict). We train the 2 models by Adam optimizer with an initial learning rate of 0.001 and a linear decaying schedule (kicking in in t=the second half of the training) to 0 for a total of 10 epochs. We find the best checkpoint using the prediction accuracy (both for the binary prediction problem and the 2-step classification problem) evaluated on the valid seen and valid unseen data of ALFRED.

Pre-trained Mask RCNN

We use a pre-trained Mask RCNN in multiple occurrences in our framework. We directly use the checkpoint provided by E.T., which is trained for predicting the instance and segmentation masks of objects in ALFRED.

2.6.6 Online Planner

During online planning, we always keep track of the agent's current position, pose, and so on (to be more precise, the (x, y, r, h) tuple). Then at each navigation subgoal, we can perform path planning using Dijkstra's algorithm given the acquired waypoints in our affordance-aware semantic representation. To decide the target horizon of the agent, we perform online exploration to cover 6 different vertical camera angles and select the one with the largest mask area for the target object type (predicted by a pre-trained model, with confidence > 0.8). We also add a backtracking mechanism in case the estimated best horizon h^* does not work for the subsequent object interaction subgoals. In specific, if the subsequent object interaction subgoal fails (i.e., the agent encounters an action failure), we find another horizon and perform inference for the object interaction subgoal again. In total, we try 3 times for $h = h^*, h^* + 15°, h^* - 15°$.

2.7 Conclusion, Limitation, and Future Work

Task decomposition is critical for acquiring AI agents to perform long-horizon tasks. It can be instantiated via two aspects presented in this work, namely decomposition into task-driven exploration and execution, the latter of which is further decomposed into subgoal-level hierarchical policies. Moreover, our work presents comprehensive empirical results that substantiate the importance of affordance-aware navigation for language-guided task completion. We propose an Affordance-aware Multimodal Neural SLAM (AMSLAM) that constructs an accurate affordance-aware semantic representation and collects data efficiently through a novel task-driven multimodal exploration module. We conduct thorough ablation studies to demonstrate that the various aspects of our design choices are essential to the performance. AMSLAM achieves more than 40% improvement over prior published work on the ALFRED benchmark and sets a new state-of-the-art generalization performance at a success rate of 23.48% on test unseen scenes.

The existing framework in our proposal regarding the affordance-aware semantic representation can be further improved by: (1) Introducing an instance-level representation such that multiple instances of the same object type can be handled better. (2) Training with more room layouts, such as those in [190], to prevent overfitting as currently there are only 108 scenes in the training data of ALFRED. AMSLAMcurrently only supports high-level actions (discrete actions via APIs with target objects specified by segmentation masks). Long-horizon tasks with low-level levels typically require skill chaining as in [54], which is largely based on online RL. We present how to tackle low-level tasks (short-horizon, though) by further adopting the idea of task decomposition in the next chapter.

Acknowledgement

Chapter 2, in full, is a reprint of the material published in the 2022 International Conference on Intelligent Robots and Systems (IROS): "Learning to Act with Affordance-Aware Multimodal Neural SLAM" (Zhiwei Jia, Kaixiang Lin, Yizhou Zhao, Qiaozi Gao, Govind Thattai, and Gaurav Sukhatme). The dissertation author was the primary investigator and author of this paper.

Chapter 3

Task Decomposition for Low-level Object Manipulation

As pointed out by Moravec's paradox, low-level control tasks are usually harder for AI agents to learn than high-level reasoning tasks (including the long-horizon tasks in ALFRED presented previously). In this section, we study challenging short-horizon contact-rich object manipulation tasks with high-precision and/or object-level geometric variations by further decomposing them into subskills (e.g., atomic actions). We develop a novel hierarchical imitation learning method that can utilize scalable, albeit sub-optimal, demonstrations. We first propose an observation space-agnostic approach that efficiently discovers the multi-step subgoal decomposition (sequences of key observations) of the demos unsupervisedly. By grouping temporarily close and functionally similar actions into subskill-level segments, the discovered breakpoints (the segment boundaries) constitute a chain of planning steps (i.e., the chain-of-thought) to complete the task. Next, we propose a Transformer-based design that effectively learns to predict the decomposition (the chain-of-thoughts) as the high-level future guidance for low-level actions by using prompt tokens and a hybrid masking strategy during training. Our model consistently surpasses existing strong baselines[1].

[1]The code is publicly available here.

3.1 Introduction

Hierarchical RL (HRL) [67] has attracted much attention in the AI community as a promising direction for sample-efficient and generalizable policy learning. HRL tackles complex sequential decision-making problems by decomposing them into simpler and smaller sub-problems via temporal abstractions (the so-called chain-of-thought [170]). In addition, many adopt a two-stage policy and possess the planning capabilities for high-level actions (i.e., subgoals or options) to achieve generalizability. On the other hand, imitation learning (IL) remains one of the most powerful approaches to training autonomous agents. Without densely labeled rewards or on-policy / online interactions, IL usually casts policy learning as (self-)supervised learning with the potential to leverage large-scale pre-collected demonstrations, usually with Transformer, as inspired by the recent success of large language models (LLMs). An obstacle in building foundational decision-making models [179] remains the better use of sub-optimal demonstrations. In this paper, we study hierarchical IL from sub-optimal demonstrations for low-level control tasks.

Despite the recent progress [22, 47, 146, 98, 2], it remains extremely challenging to solve low-level control tasks such as contact-rich object manipulations by IL in a scalable manner. Usually, the demonstrations are inherently sub-optimal because of the underlying contact dynamics [120] and the way they are produced. The undesirable properties, such as being non-Markovian, noisy, discontinuous, and random, pose great challenges in both the optimization and the generalization of the imitators (see detailed discussion in Sec. 3.4). We find that, by adopting the hierarchical principles (i.e., temporal abstraction and high-level planning) into our Transformer-based design, we can enjoy large-scale (albeit sub-optimal) demonstrations for their performance boost on solving challenging tasks. To achieve this, we first propose an unsupervised chain-of-thought discovery strategy to generate CoT supervision from the demonstrations. We then design our model to learn to dynamically generate CoT guidance for better low-level action predictions.

Specifically, we consider the multi-step subgoal decomposition of a task into a chain of planning steps as its chain-of-thought (inspired by CoT [170] and PC [178]). As part one of our contribution, we propose an observation space-agnostic approach that efficiently discovers the chain-of-thought (CoT), defined as a sequence of key observations, of the demos in an unsupervised manner. We propose to group temporarily close and functionally similar actions into subskill-level segments. Then the breakpoints (the segment boundaries) naturally constitute the CoT that represents the high-level task completion process. For part two, we propose a novel Transformer-based design that effectively learns to predict the CoT jointly with the low-level actions. This coupled prediction mechanism is achieved by adding additional prompt tokens at the beginning of the context history and by adopting a hybrid masking strategy. As a result, CoT guidance is dynamically updated at each step and better feature representation of the trajectories is learned, eventually improving the generalizability of the low-level action prediction process.

We call our method Chain-of-Thought Predictive Control (CoTPC). From an optimization perspective, it learns faster from sub-optimal demos by utilizing the subgoals (CoTs) that are usually more robust and admit less variance. From a generalization perspective, it uses Transformers [12] to improve generalization with CoT planning, which is learned from the unsupervisedly discovered CoTs from the demos. We evaluate CoTPC on several challenging low-level control tasks (Moving Maze, Franka-Kitch, and ManiSkill2) and verify its design with ablation studies.

3.2 Related Work

Learning from Demonstrations (LfD)

Learning interactive agents from pre-collected demos has been popular due to its effectiveness and scalability. Roughly speaking, there are three categories: offline RL, online RL with auxiliary demos, and behavior cloning (BC). While offline RL approaches

[89, 49, 94, 90, 88, 22, 169] usually require demonstration with densely labeled rewards and the methods that augment online RL with demos [61, 82, 137, 109, 131, 62, 119, 151] rely on on-policy interactions, BC [121] formulates fully supervised or self-supervised learning problems with better practicality and is adopted widely, especially in robotics [184, 47, 125, 187, 11, 127, 48, 185]. However, a well-known shortcoming of BC is the compounding error [137], usually caused by the distribution shift between the demo and the test-time trajectories. Various methods were proposed to tackle it [136, 137, 157, 93, 164, 10, 16]. Other issues include non-Markovity [103], discontinuity [47], randomness and noisiness [139, 173] of the demos that results in great compounding errors of neural policies during inference (see Sec. 3.4 for detailed discussions).

LfD as Sequence Modeling

A recent trend in offline policy learning is to relax the Markovian assumption of policies, partially due to the widespread success of sequence models [53, 29, 167] where model expressiveness and capacity are preferred over algorithmic sophistication. Among these, [38, 102] study one-shot imitation learning, [101, 151] explore behavior priors from demos, [22, 98, 73, 146, 2, 72] examine different modeling strategies for policy learning. In particular, methods based on Transformers [167, 12] are extremely popular due to their simplicity and effectiveness.

Hierarchical Approaches in Sequence Modeling and RL

Chain-of-Thought [170] refers to the general strategy of solving multi-step problems by decomposing them into a sequence of intermediate steps. It has recently been applied extensively in a variety of problems such as mathematical reasoning [97, 33], program execution [135, 111], commonsense or general reasoning [130, 30, 96, 170], and robotics [176, 189, 76, 54, 178, 148, 70]. Similar ideas in the context of HRL can date back to Feudal RL [39] and the option framework [158]. Inspired by these approaches, ours focuses on the imitation learning setup (without reward labels or online interactions) for low-level control tasks. Note that while Procedure cloning [178] shares a similar name to our paper, it suffers from certain limitations that make it much less applicable (see detailed discussion in Sec. 3.2.1).

Demonstrations for Robotics Tasks

In practice, the optimality assumption of the demos is usually violated for robotics tasks. Demos involving low-level actions primarily come in three forms: human demo captured via teleoperation [91, 168], expert demo generated by RL agents [106, 107, 25, 75], or those found by planners (e.g., heuristics, sampling, search) [55, 126, 45]. These demos are in general sub-optimal due to either human bias, imperfect RL agents, or the nature of the planners. In this paper, we mostly focus on learning from demos generated by planners in ManiSkill2 [55], a benchmark not currently saturated for IL while being adequately challenging (see details in Sec. 3.6.3).

3.2.1 Extended Discussions of Closely Related Work

Procedure Cloning (PC)

PC [178] was recently proposed to use intermediate computation outputs of demonstrators as additional supervision for improving the generalization of BC policies. However, it assumes full knowledge of the demonstrators, including the usually hidden computations that consist of potentially large amounts of intermediate results. For instance, in the graph search example used in the original paper, PC requires knowing the traversed paths of the BFS algorithm, such as the status of each node, either the included ones or *the rejected (and so omitted) ones* in the final returned result. Whereas, CoTPC does not require such knowledge, as the CoTs are included as part of the results, not hidden intermediate computations, and the CoT supervision itself can be obtained via the unsupervised discovery method. Moreover, machine-generated demonstrations can be crowd-sourced and the demonstrators are usually viewed as black boxes, making this a limitation of PC.

Policy Learning with Motion Planning

Some existing work [148, 70] adopt a strategy similar to CoTPC in terms of predicting key states (the waypoints) as high-level policies. However, they use motion planners as low-level policies, while CoTPC directly learns to predict low-level controls. The major advantage of our approach is to handle environments requiring dynamic (or reactive) controls (like Moving Maze and Push Chair), where the key states must be updated at every step and so motion planner-based strategies will struggle. Also, PerAct [148] is relatively limited due to its discretized actions, especially for tasks requiring high-precision manipulation (e.g., Peg Insertion). Moreover, some existing work [126, 46] uses motion planners as the only demonstrators to acquire neural policies, which are to some extent constrained to tasks involving quasi-static control.

Robotic Transformer-1 (RT-1)

RT-1 [11] is a concurrent work that also directly models low-level control actions with a Transformer. It benefits from the sheer scale of real-world robot demonstration data pre-collected over 17 months and the tokenization of both visual inputs (RGB images) and low-level actions. While RT-1 shows great promise in developing decision foundation models for robotics, it adopts the conventional auto-regressive Transformer without explicitly leveraging the structural knowledge presented in low-level control tasks. Moreover, it is so computationally intensive that it usually only admits less than 5 control signals per second. Our work, CoTPC, is an early exploration in this direction and we believe it will inspire the future designs of generally applicable models for robotics tasks. Another difference is that since RT-1 discretizes the action space, it might suffer from degraded performance for tasks that require high precision (such as Peg Insertion).

3.3 Preliminaries

MDP Formulation

One of the most common ways to formulate a sequential decision-making problem is via a Markov Decision Process, or MDP [65], defined as a 6-tuple $\langle S, A, \mathcal{T}, \mathcal{R}, \rho_0, \gamma \rangle$, with a state space S, an action space A, a Markovian transition probability $\mathcal{T} : S \times A \to \Delta(S)$, a reward function $\mathcal{R} : S \times A \to \mathbb{R}$, an initial state distribution ρ_0, and a discount factor $\gamma \in [0, 1]$. An agent interacts with the environment characterized by $\mathcal{T}$ and $\mathcal{R}$ according to a policy $\pi : S \to \Delta(A)$. We denote a trajectory as τ_π as a sequence of $(s_0, a_0, s_1, a_0, ..., s_t, a_t)$ by taking actions according to a policy π. At each time step, the agent receives a reward signal $r_t \sim \mathcal{R}(s_t, a_t)$. The distribution of trajectories induced by π is denoted as $P(\tau_\pi)$. The goal is to find the optimal policy π^* that maximizes the expected return $\mathbb{E}_{\tau\sim\pi}[\sum_t \gamma^t r_t]$. Notice that, in robotics tasks and many real-world applications, the reward is at best only sparsely given (e.g., a binary success signal) or given only after the trajectory ends (non-Markovian).

Behavior Cloning

The most straightforward approach in IL is BC, which assumes access to pre-collected demos $D = \{(s_t, a_t)\}_{t=1}^N$ generated by expert policies and learns the optimal policy with direct supervision by minimizing the BC loss $\mathbb{E}_{(s,a)\sim D}[-\log \pi(a|s)]$ w.r.t. a mapping π. It requires the learned policy to generalize to states unseen in the demos since the distribution $P(\tau_\pi)$ will be different from the demo one $P(\tau_D)$ at test time, a challenge known as distribution shift [136]. Recently, several methods, particularly those based on Transformers, are proposed to relax the Markovian assumption. Instead of $\pi(a_t|s_t)$, the policy represents $\pi(a_t|s_{t-1}, s_{t-2}, ..., s_{t-T})$ or $\pi(a_t|s_{t-1}, a_{t-1}, ..., s_{t-T}, a_{t-T})$, i.e., considers the history up to a context size T. This change was empirically shown to be advantageous.

3.4 Challenges of Learning from Sub-optimal Demos for Low-level Control

There are several technical challenges that make imitation learning or offline RL difficult from sub-optimal demonstrations, especially for low-level control tasks. In this section, we briefly discuss four of them.

Non-Markovity

While each trajectory in the demos can be represented by a Markovian policy, the Markovian policy linearly combined from them by perfectly imitating the combined demos can suffer from a negative synergic effect if there are conflicts across demos. This is because the demos might be generated by different agents or different runs of the same algorithm. It becomes even worse when the demonstrations themselves are generated by non-Markovian agents (e.g., human or planning-based algorithms). Instead, a non-Markovian policy is more universal and can resolve conflicts by including history as an additional context to distinguish between different demos.

Noisiness

Sometimes the demo trajectories are intrinsically noisy with divergent actions produced given the same states. For instance, a search-based planner returns more than one possible action given the same action and state history to solve the task. At times, the demo actions are even distributed uniformly (e.g., with motion planning algorithms as demonstrators). This leads to increased uncertainty and variance of the cloned policies and so higher compounding errors. Note that multi-modality is a related but orthogonal issue [146], i.e., when an unimodal estimate of the (continuous) action distribution leads to a significantly worse return.

Discontinuity

For low-level control tasks, demo policies often consist of sharp value changes or topology changes (e.g., due to contact changes). Such discontinuity in the underlying state-to-action mapping leads to difficulties in learning a robust and accurate model, thus harming generalizability. A recent method [47] deals with this by an energy-based implicit model in place of an explicit one. While theoretically sound, it is shown [146] to be less practical for non-Markovian implicit models, and several later non-Markovian explicit models outperform it.

Randomness

The actual or apparent unpredictability usually exists in sub-optimal demonstrations either because the intermediate computations of the demonstrators are not revealed in the demos (e.g., the shortest paths generated by BFS do not reveal the intermediate search process), or the demonstrators are inherently non-deterministic (e.g., relying on rejection sampling). Such a trait makes IL less robust as the decision-making patterns from demos might be unclear, hard to learn, and not generalizable [116]. For instance, in a continuous action space maze, a solution found by random search is more-or-less a winning lottery ticket, whose pattern might not be very generalizable.

3.5 Method

To develop a scalable and powerful imitation learning algorithm for diverse yet sub-optimal demonstrations, especially for low-level control tasks, we propose (1) an observation space-agnostic strategy that efficiently discovers the subgoal sequences from the demonstrations (as chain-of-thought supervision) in an unsupervised manner and (2) a novel Transformer-based design that effectively adopts the hierarchical principles, the CoT planning (i.e., subgoal sequence predictions), in imitation learning.

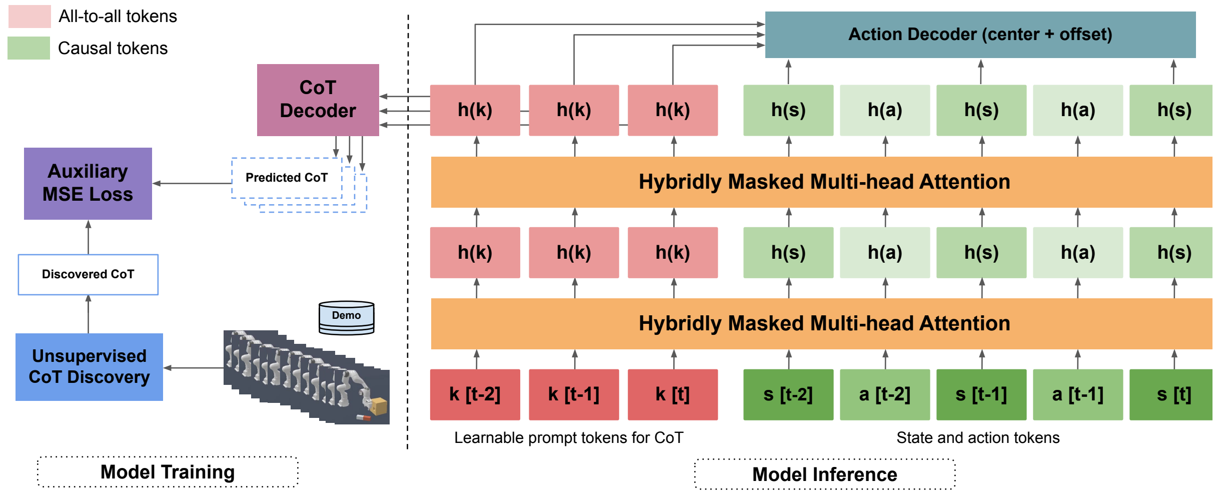


Figure 3.1. During training, CoTPC learns to jointly predict (1) the next & the last subgoals from each CoT token and (2) the low-level actions from each state token. During inference, without the CoT decoder, the low-level actions are predicted with the center & offset predictors from different tokens (detailed in Sec. 3.5.2) with the guidance of the dynamically updated CoT features. The CoT tokens are all-to-all (can see any tokens). The state and action tokens are causal (can only see previous and CoT tokens). Only 2 attention layers and 3 timesteps are shown for better display.

3.5.1 Unsupervised Discovery of Chain-of-Thought from Demos

We observe that many low-level control tasks (e.g., object manipulations) naturally consist of sequences of subgoals. In a succeeded trajectory, there exist key observations, each of which marks the completion of a subgoal. For instance, in Moving Maze illustrated in Fig. 3.4, the two bridges naturally divide the task into three subgoals. We denote such a multi-step subgoal decomposition of a task into a chain of planning steps as its chain-of-thought. The CoT provides coherent and succinct behavior guidance - typical benefits of hierarchical policy learning. Formally, for each trajectory $\tau \in D$, we define CoT as a sequence of its observations $F_{cot}(\tau) = \{s_t | s_t \in \tau\} = \{s_k^{\text{cot}}\}$. In this section, we present an intuitive strategy to discover CoT from demos as the CoT supervision for our model introduced shortly. Our unsupervised approach is observation space-agnostic and relies on neither human supervision nor reward design.

First of all, each demo trajectory can be considered as an execution of a chain of subskills to complete the corresponding sequence of subgoals. Accordingly, the ending observations of each subskill constitute the subgoal sequence (i.e., chain-of-thought). Also, note that the number of these subskills is not known *a prioi*, since demonstrations of the same task can still have variability in terms of the starting configurations and execution difficulty. Our base assumption is that actions within the same subskill are temporally close and functionally similar. We, therefore, propose to *group contiguous actions into segments*, using a similarity-based heuristic to find these subskills. We provide visualization of similarity maps across actions in the same trajectory in Fig. 3.2 to illustrate that there naturally exists such grouping patterns by functional similarities. While an action sequence is modeled as a time series, its subgoals are the key observations corresponding to the changepoints of the time series. We then utilize the Pruned Exact Linear Time (PELT) method [83] with cosine similarity as the cost metric to generate the changepoints in a per-trajectory manner. This unsupervised formulation has one key hyperparameter β which controls the penalty for adding more changepoints thereby determining the number of subgoals (i.e., the length of the CoT). In our experiments, we select β based on a small set of validation data and we find it relatively robust.

We find our approach discovers meaningful subskills (the action segments) for a diverse set of tasks with different action spaces, based on a qualitative evaluation of the key observations at the discovered changepoints (i.e., the chain-of-thought). For instance, the execution of Peg Insertion consists of reaching, grasping, aligning the peg with the hole, micro-adjusting, and steady insertion of the peg. We illustrate some CoT discovery results in Fig. 3.3. Specifically, we visualize the sub-stages (and thus the subgoals) extracted from two trajectories using the automatic CoT discovery process described previously.

Besides being unsupervised, another major benefit of our approach is that it is observation space-agnostic with the discovery invariant of the specific sensor setup (e.g., camera angles). This eliminates the need to manually tune the observation space (e.g.,

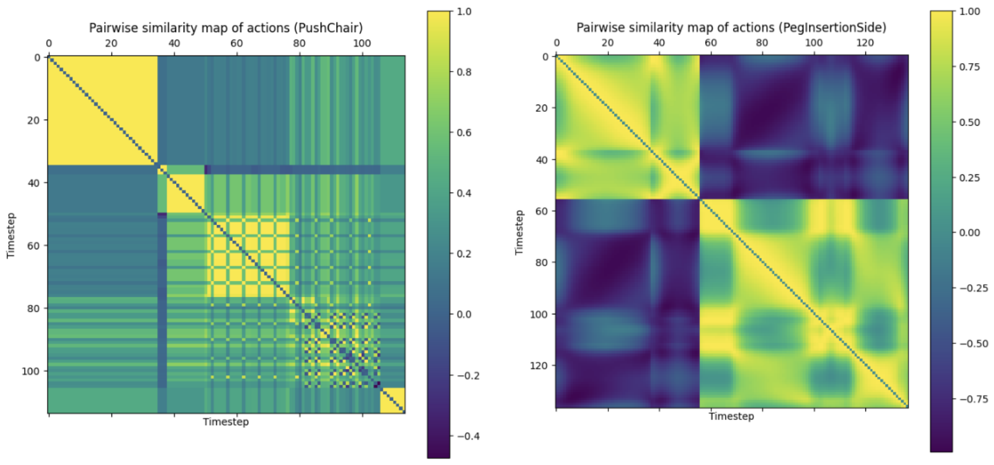


Figure 3.2. Pairwise similarities of actions at different timesteps in two trajectories for two example tasks, Push Chair (**left**) and Peg Insertion (**right**). Note that the action spaces for these tasks are distinct: the former uses delta joint velocity control and the latter uses delta joint pose control. Visually identifiable blocks along the diagonal, which have high correlation values among most of its members, tend to correspond very well with human intuition of subskills (i.e. similar actions at nearby timesteps belong to the same subskill). We find that this strategy allows for automatic, scalable, and meaningful subgoal discovery in an unsupervised manner.

the camera setups) to detect the key observations.

3.5.2 Chain-of-Thought Guided Action Modeling

In this section, we introduce our Transformer-based design that learns to model the CoTs and predict the actions accordingly based on the supervision provided by the aforementioned CoT discovery.

Learnable Prompt Tokens for CoT with Hybrid Masking

We base our architecture on the recently proposed Behavior Transformer (BeT) [146], which empowers GPT [12] with a discrete center plus continuous offsets prediction strategy for modeling diverse and noisy action sequences. To predict CoTs, we propose to add a set of learnable prompt tokens [191] at the beginning of the state and action context history. We train these tokens to extract features from the sequence to predict CoTs together with a CoT decoder (similar to the object query tokens in Detection Transformer

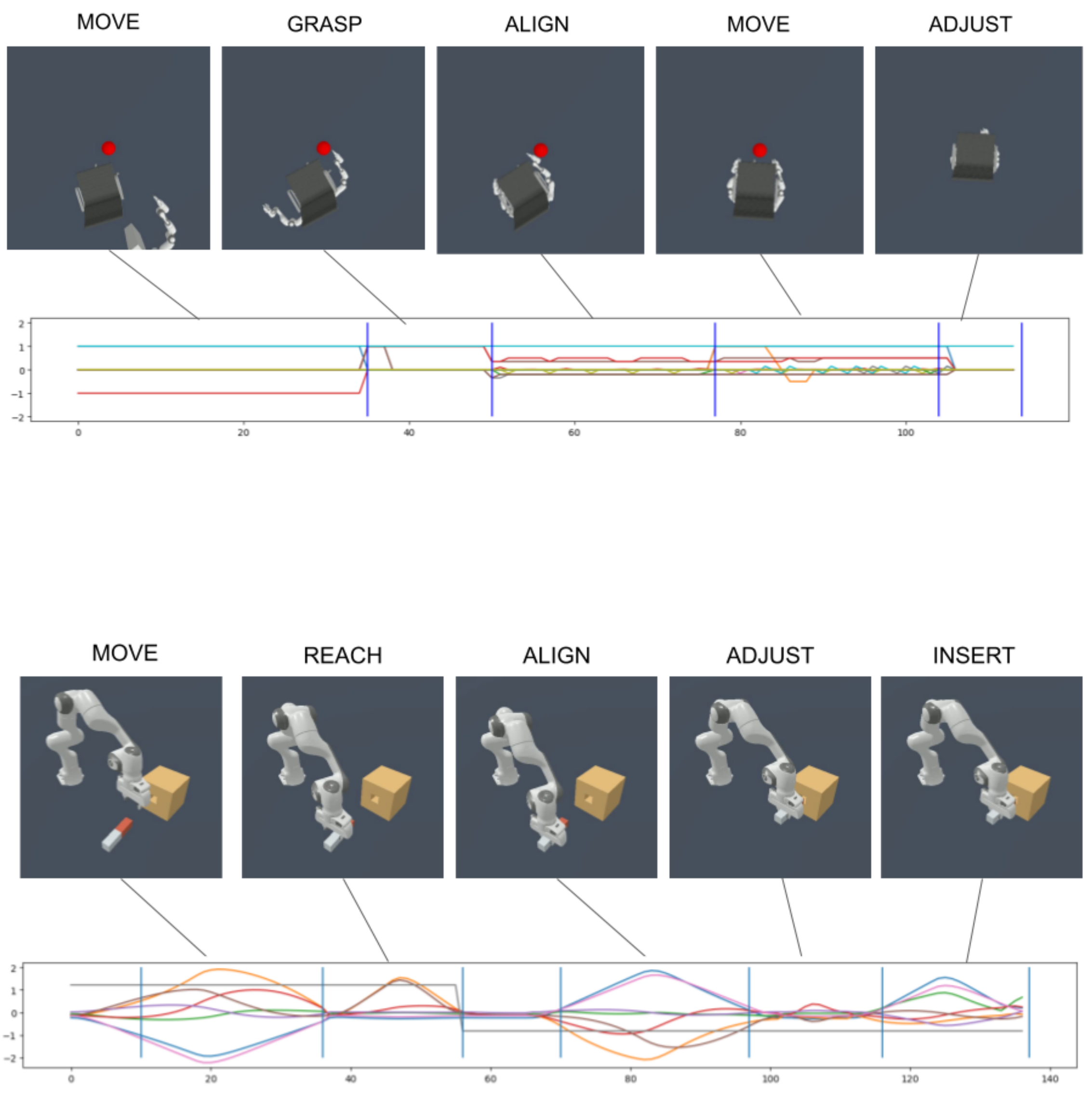


Figure 3.3. Illustration of actions corresponding to different stages and the associated observations for two tasks: Push Chair (**top**) and Peg Insertion (**bottom**). The stages are discovered by grouping the actions into subskills by our unsupervised CoT discovery method.

[15]). We design a *hybrid* masking regime, where during inference, the CoT tokens are all-to-all and can observe all action and state tokens in the context history, and the state or action tokens attend to those in the past (standard causal mask) including the CoT tokens. In this way, the action decoding is guided by the extracted CoT features. Formally, given a context size of T, let us denote the CoT tokens as $\{\mathbf{S}^{\mathbf{cot}}_{...}\}$. We model the demo trajectory segment of length T up to timestep t, i.e., $\tau_T(t) = \{\mathbf{S}^{\mathbf{cot}}_{...}, s_{t-(T-1)}, a_{t-(T-1)}, ..., s_{t-1}, a_{t-1}, s_t\}$ by applying the hybridly masked multi-head attention, denoted $\mathrm{MHA}_{hmask}[\cdot]$. Features from a total of J attention layers are

$$h_j(\tau_T(t)) = \mathrm{MHA}_{hmask}[F_{enc}(\tau_T(t))], \quad j = 1$$
$$h_j(\tau_T(t)) = \mathrm{MHA}_{hmask}[h_{j-1}(\tau_T(t))], \quad j > 1$$

where F_{enc} encodes each action token and state token by encoder $f_a(\cdot)$ and $f_s(\cdot)$, respectively (no encoder for the CoT tokens). Here we omit the position embeddings and the additional operations between the attention layers as in standard Transformers. Note that we also put the action sequences into the context in our variant of BeT and our model as we find this generally performs better.

Coupled Action and CoT Predictions with Shared Tokens

BeT adopts a pair of centers and offsets predictors, $g_a^{\mathbf{ctr}}(\cdot)$ and $g_a^{\mathbf{off}}(\cdot)$, with features from the last attention layer corresponding to the state tokens as inputs, which are denoted as $h_J(\tau_T(t))$`[-(2T-1)::2]` (with the Python slicing notation). A vanilla design of our approach is to adopt a CoT decoder $g_{cot}(\cdot)$ taking $h_J(\tau_T(t))$`[:-(2T-1)]` as inputs, where we only need one CoT token. However, we find that a more coupled strategy produces policies much more generalizable. Specifically, we force the offset predictor of actions $g_a^{\mathbf{off}}(\cdot)$ to take the features from the CoT tokens rather than the state tokens as inputs. As $g_a^{\mathbf{off}}(\cdot)$ and $g_{cot}(\cdot)$ share the same inputs, this provides stronger CoT guidance for action modeling. To distinguish between actions predicted for different states in the sequence, we need T such CoT tokens, denoted $\{\mathbf{S}_t^{\mathbf{cot}}\}$, each of which corresponds to one state token $\{s_t\}$ (see Fig. 3.1). The features from the CoT tokens are used by the shared CoT decoder $g_{cot}(\cdot)$ to predict the CoT output (introduced shortly). An even more coupled prediction approach is to force the center predictor $g_a^{\mathbf{ctr}}(\cdot)$ to also share the inputs with $g_a^{\mathbf{off}}(\cdot)$ and $g_{cot}(\cdot)$. However, this leads to optimization challenges and instabilities as CoT tokens hold direct responsibilities for all three predictions. See ablation studies for further discussions.

We formulate an autoregressive prediction strategy for CoT by training $g_{cot}(\cdot)$ to decode the next subgoal (i.e., the first $s_k^{\mathbf{cot}}$ with $k > t$ for the demo segment $\tau_T(t)$) and the very last subgoal (usually the end of τ) from every CoT token. The predictions are denoted $g_{cot}(\mathbf{S}_t^{\mathbf{cot}})$ for each timestep t. We find this strategy outperforms both predicting the individual one, as it performs both immediate and long-term planning. The flexible numbers of subgoals across different demo trajectories necessitate the autoregressive decoding of CoTs and make our approach resonate with CoTs in LLMs, where similarly the outputs are generated jointly with the reasoning chain. Note that during inference the CoT predictor is not used and only the CoT features are. As CoT features are updated dynamically, our approach can deal with tasks involving dynamic controls (e.g., Moving Maze and Push Chair).

Model Training

The overall training pipeline is illustrated in Fig. 3.1. The model is trained with behavior cloning loss as well as the auxiliary CoT prediction loss $\mathcal{L}_{cot}$ based on MSE (weighted by a coefficient λ), which yields the overall training objective:

$$\mathcal{L}_{total} = \mathop{\mathbb{E}}_{(s_t,a_t)\in D} \mathcal{L}_{bc}(a_t, \hat{a}_t) \ + \ \mathop{\mathbb{E}}_{\tau'\in D} \frac{1}{T}\sum_t \mathcal{L}_{cot}([s_{next}^{\mathbf{cot}}(\tau'), s_{last}^{\mathbf{cot}}(\tau')], g_{cot}(\mathbf{S}_t^{\mathbf{cot}}))$$

Where $\hat{a}_t$ is the predicted action via $g_a^{\mathbf{ctr}}(\cdot)$, $g_a^{\mathbf{off}}(\cdot)$. τ' are randomly sampled segments of the demo trajectories of length T. $s_{next}^{\mathbf{cot}}(\tau')$, $s_{last}^{\mathbf{cot}}(\tau')$ are the next and last subgoal, respectively. Note that there are T CoT tokens and so there are T CoT loss terms.

During training, we apply random attention masks to the action and state tokens so that the CoT tokens attend to a context of varied length (from the first state token in the context to a randomized t-th state token with $1 <= t <= T$). In doing so, we can (1) prevent CoT tokens from directly copying the corresponding state tokens which makes offset prediction of actions trivial and (2) perform a way of data augmentation (applying a form of dropout on the attention masks).

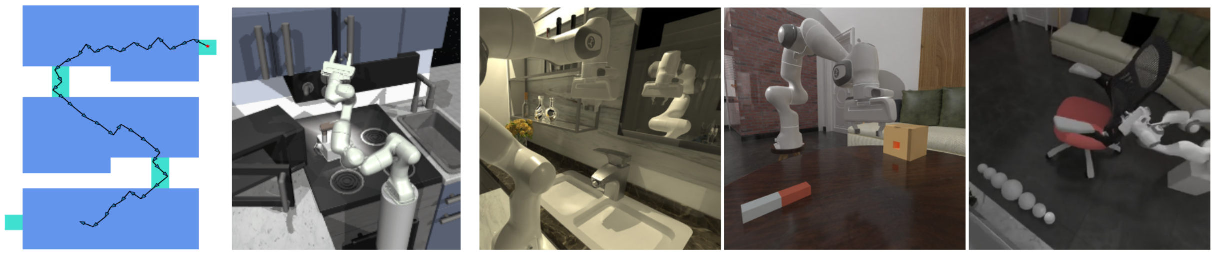


Figure 3.4. Illustration of the Moving Maze (**left**), Franka-Kitchen (**middle**) and some sampled tasks from ManiSkill2 (**right**), namely Turn Faucet, Peg Insertion and Push Chair. See detailed descriptions in Sec. 3.6.1, 3.6.2 and 3.6.3, respectively.

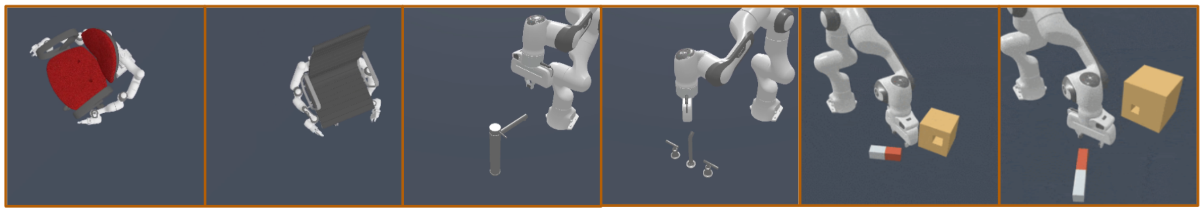


Figure 3.5. Sampled geometric variations for Push Chair, Turn Faucet and Peg Insertion. Note that the sizes of peg & box and the relative locations of the hole vary across different env. configs.

3.6 Experiments

In this section, we present experimental results for several tasks as well as for the ablation studies. While existing benchmarks are mostly saturated for IL (DMControl [161], D4RL [49], etc.) or lack demo data (e.g., MineDojo [42]), we choose a diverse range of tasks that lie in between. We first examine our approach using a 2D continuous-space Moving Maze and a variant of Franka-Kitchen [57]. We then perform extensive comparisons with 5 object manipulation tasks (ranging from relatively easy ones to very challenging ones) from ManiSkill2 [55]. These tasks are of several categories (navigation, static/mobile manipulation and soft-body manipulation, etc.), various action spaces (delta joint pose, delta velocity, etc.), with different sources of the demos (human, heuristics, etc.) and with different reasons for being challenging (object variations, long-horizon, etc.).

3.6.1 Moving Maze

We present a 2D maze with a continuous action space of displacement $(\delta x, \delta y)$. As shown in Fig. 3.4 (left), in this s-shaped maze, the agent starts from a location randomly initialized inside the top right square region (in green) and the goal is to reach the bottom left one (also in green). Upon each environment reset, the two regions as well as the two rectangular bridges (in green) have their positions randomized. During the game, each of them except for the top square moves (independently) back and forth with a randomized constant speed. Once the agent lands on a moving block, the block will immediately become static. The agent cannot cross the borders of the maze (but it will not die from doing so). For simplicity, we adopt a 10-dim state observation consisting of the current location of the agent and the four green regions. This task requires *dynamic control/planning.*

Demonstrations

To enable policy learning from demonstrations, we curate demo trajectories (each with a different randomized environment configuration) by adopting a mixture of heuristics and an RRT-style planner with hindsight knowledge not available at test time (see details in Sec. 3.7.2). This setup follows recent work [55] for leveraging machine-generated demonstrations.

Training and Evaluation

For this task, we compare CoTPC with vanilla BC, Behavior Transformer (BeT) [146], Decision Transformer (DT) [22]. DT was originally proposed for offline RL with demonstrations of dense rewards. We adapt it for the BC setup by ignoring the reward tokens. We add action tokens to BeT (like in DT) and build CoTPC on top of BeT. We implement CoTPC, DT, and BeT with the shared Transformer configuration for a fair comparison. We train all methods on 400 demo trajectories of different env. configs and evaluate on 100 unseen ones (results in Tab. 3.1).

3.6.2 Franka Kitchen

Different variants and task setups of the Franka Kitchen environment have been studied previously [57, 139, 49]. We propose a setting where the agent is asked to complete 4 object manipulations (out of 7 different options) in an order specified by the goal. We use a strict criterion, i.e., the task succeeds when all 4 sub-tasks are completed. The sub-tasks need to be done in the requested order to be counted as completed. The environment will terminate when a sub-task other than the specified 4 is performed. The action space is based on the joint velocity (8-dim) of the robot. We use the original state observation appended with the modified goal embedding.

Table 3.1. Test performance on Moving Maze and (a variant of) Franka Kitchen. SR (%) is the task success rate (for Franka Kitchen it means completion of all 4 sub-tasks). # s-tasks means the avg. number of completed sub-tasks per trajectory rollout. The best results are **bolded**.

	Vanilla BC	DT	BeT	CoTPC (ours)
Moving Maze (SR)	9.0	23.0	33.0	**44.0**
Franka Kitchen (#s-tasks/SR)	1.7/6.7	1.6/6.7	1.8/14.4	**2.1/25.6**

Demonstrations

We replay a subset of the human demonstrations originally proposed in [57]. Specifically, we use 50 demo trajectories of length ranging from 150 to 300 and relabel them with what sub-tasks are performed and in what order, for each of the trajectories. As a result, many ordered sub-task combinations admit at most one demo trajectory. See more details in Sec. 3.7.2.

Training and Evaluation

We use the same set of baselines as in Moving Maze. We evaluate using 90 unseen env. configs, which vary in initial scene configs (all ordered sub-task combinations have been observed in the demo, though). This task requires generalizable IL due to the limited amount of human demos and the diverse set of ordered sub-task sequences. Also see results in Tab. 3.1.

3.6.3 ManiSkill2

ManiSkill2 [55] is a recently proposed extension of ManiSkill [107], which features a variety of low-level object manipulation tasks in environments with realistic physical simulation (e.g., fully dynamic grasping motions). We choose 5 tasks (some illustrated in Fig. 3.4). Namely, Stack Cube for picking up a cube, placing it on top of another, and the gripper leaving the stack; Turn Faucet for turning on different faucets; Peg Insertion for inserting a cuboid-shaped peg *sideways* into a hole in a box of different geometries

Table 3.2. Test performance (success rate) on the unseen and the 0-shot setup for ManiSkill2 tasks with state observations. The best results are **bolded**. Note that Pour does not support state observations.

	Stack Cube	Peg Insertion	Turn Faucet		Push Chair	
	unseen	0-shot	unseen	0-shot	unseen	0-shot
Vanilla BC	1.0	0.0	0.0	0.0	0.0	0.0
Decision Transformer	19.0	17.5	40.0	27.0	25.6	17.0
Decision Diffuser	26.0	12.6	17.0	5.0	**56.0**	20.0
Behavior Transformer	73.0	42.5	49.6	32.5	44.0	33.4
CoTPC (ours)	**86.0**	**59.3**	**50.0**	**39.3**	51.2	**41.0**

Table 3.3. Test performance (success rate) on the unseen and the 0-shot setup for ManiSkill2 tasks for point cloud observations. The best results are **bolded**. We only show the best baseline here, i.e., BeT.

	Cube	Peg	Pour	Faucet		Chair	
	unseen	0-shot	unseen	unseen	0-shot	unseen	0-shot
Behavior Transformer	70.0	35.0	24.0	50.0	20.0	26.0	13.4
CoTPC (ours)	**81.0**	**44.0**	**32.0**	**58.0**	**27.5**	**32.0**	**16.7**

and sizes; Push Chair for pushing different chair models into a specified goal location (via a mobile robot); and Pour for pouring liquid from a bottle into the target beaker with a specified liquid level. Push Chair adopts a delta joint velocity control (19-dim, dual arms with mobile base); Pour adopts delta end effector pose control (8-dim); the rest uses delta joint pose control (8-dim). We perform experiments with both state and point cloud observations.

Task Complexity

The challenges of these tasks come from several aspects. Firstly, all tasks have all object poses fully randomized (displacement around 0.3m and 360° rotation) upon environment reset (this is in contrast to environments such as Franka Kitchen). Secondly, Turn Faucet, Peg Insertion, and Push Chair all have large variations in the geometries and sizes of the target objects (see illustrations in Fig. 3.5). Moreover, the faucets are mostly pushed rather than grasped during manipulation (under-actuated control), the holes have 3mm clearance (requiring high-precision control) and it needs at least half of the peg to be pushed sideway into the holes (harder than similar tasks in other benchmarks [177]), the chair models are fully articulated with lots of joints, and the pouring task requires smooth manipulation without spilling the liquid. Moreover, ManiSkill2 adopts impedance controllers that admit smoother paths than the position-based (or specifically-tuned) ones while at the cost of harder low-level action modeling (e.g., actuators can be quite laggy).

Demonstrations

The complexity of the tasks also lies in the sub-optimality of the demos (e.g., vanilla BC struggles on all 5 tasks). The demos here are generated by a mixture of multi-stage motion-planing and heuristics-based policies (with the help of privileged information in simulators). We use 500 demo trajectories for Stack Cube and Turn Faucet (distributed over 10 faucets), 1000 demos for Peg Insertion and Push Chair (distributed over 5 chairs), and 150 demos for Pour.

Training and Evaluation

Besides vanilla BC, DT, and BeT, we add Decision Diffuser (DD) [2] as a baseline, which explores the diffusion model [64] for policy learning by first generating a state-only trajectory and then predicting actions with an acquired inverse dynamics model. As tasks in ManiSkill2 feature diverse object-level variations, they provide good insights into both how effective an imitator can learn the underlying behavior and how generalizable it is. We evaluate using the 5 tasks in both unseen (seen objects but unseen scene configs) and 0-shot (unseen object geometries) setup. Specifically, we have all but Peg Insertion with the unseen setup and Turn Faucet, Push Chair & Peg Insertion with the 0-shot setup. We use task success rate (SR) as the metric. Results are reported in Tab. 3.2

Table 3.4. Results from the ablation studies (unseen SR for Push Chair and 0-shot SR for Peg Insertion).

	DECOUPLED	ONLY LAST	ONLY NEXT	RANDOM	VANILLA	O-SHARED	SWAPPED	CoTPC
PEG INSERTION	47.0	52.0	49.0	41.0	45.0	39.0	46.0	59.3
PUSH CHAIR	36.0	36.0	37.0	31.0	35.0	29.0	32.0	41.0

for state observation and Tab. 3.3 for point cloud observations, where we demonstrate the clear advantages of CoTPC. Note that we only select the best baseline (i.e., BeT) for experiments with point cloud observations, where we use a lightweight PointNet [123] for encoding the point clouds. For the main results of all methods on ManiSkill2, we report the best performance among 3 training runs over the last 20 checkpoints, an eval protocol adopted by [27].

3.6.4 Ablation Studies

We present ablation studies on two tasks (Peg Insertion and Push Chair) from ManiSkill2 (we use state observations). We summarize the results in Tab. 3.4 and introduce the details below.

Decoupled prediction of CoT and actions

In this variant denoted `decoupled`, we train another Transformer (denoted CoT Transformer) with the same state and action sequence as inputs to predict the CoT (the next and the last subgoal). While the original Transformer learns to only predict the actions with the discovered CoT fed into the CoT tokens. During inference, the CoT predicted by the CoT Transformer is fed instead. This decoupled CoT and action prediction strategy is inferior to the coupled one since the latter not only learns to leverage predicted CoT as guidance but also encourages better feature representation of the trajectory by sharing the features for these tasks.

What subgoals to predict from the CoT tokens?

In the variant named `only last`, we ask the CoT decoder $g_{cot}(\cdot)$ to only predict the last subgoal. In the variant named `only next`, we ask it to only predict the next subgoal. We find that predicting both works the best as it provides the model with both immediate and long-term planning. In the variant named `random`, we ask it to predict a single randomly selected observation from the future as a random subgoal (not the ones from our CoT discovery). This leads to the worst results as the action guidance is not very helpful.

Shared tokens for CoT and action predictions

A vanilla design of jointly predicting CoT and actions is to only use one CoT token with the center and offset predictors on top of the state tokens and the CoT decoder on top of the CoT token, denoted as `vanilla`. We further force both the center and the offset predictors to take CoT tokens as inputs, denoted `o-shared`. This variant overly shares the CoT tokens for all predictors/decoders and suffers from optimization instabilities. In the last variant, denoted `swapped`, we let the offset predictor take the action tokens as inputs and the center predictor take the CoT tokens. We find this alternative setup performs worse.

3.6.5 Preliminary Results of Sim-to-Real Transfer

We examine the plausibilities of sim-to-real transfer of our state-based CoTPC in a zero-shot setup on two tasks, namely, Stack Cube and Peg Insertion. Our real-world experiment setup and two sampled succeeded trajectories are illustrated in Fig. 3.6. With an off-the-shelf pose estimation framework such as PVNet [118], we can achieve reasonable performance using the state-based CoTPC policy learned purely from simulated data. As a preliminary examination, we only perform qualitative evaluations. See detailed camera setup in [23].

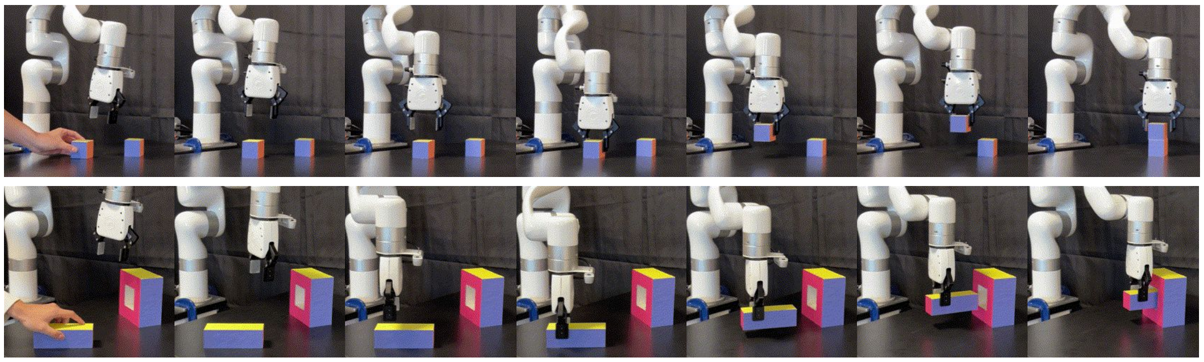


Figure 3.6. Two sampled succeeded trajectories for Stack Cube and Peg Insertion, respectively, in a real robot setup, from state-based CoTPC policies trained purely from demos in simulators. As an early examination, we increase the clearance for peg insertion from 3mm (sim) to 10mm (real) and only use peg and box-with-hole models of fixed geometry.

3.7 Implementation Details

3.7.1 Details of the Environments

Moving Maze

Moving Maze is a 2D maze with a continuous action space of displacement $(\delta x, \delta y)$, where both components $\in [1.5, 4]$. This an s-shaped maze whose height is 80 and width is 60 with the agent starting from a location randomly initialized inside the top right square region (in green) and the goal is to reach the bottom left one (also in green). Upon each environment reset, the two regions (the starting square and the target square) as well as the two rectangular bridges (in green) have their positions randomized. Specifically, The two square regions are randomized to the right of the top and to the left of the bottom blue islands, respectively. Their initial locations vary with a range of 20 vertically. The two bridges' initial locations vary also with a range of 20 horizontally. During the game, each of them except for the top square moves (independently) back and forth with a randomized constant speed $\in [1, 2]$. Once the agent lands on a moving block, the block will immediately become static. The agent cannot cross the borders of the maze (but it will not die from doing so). For simplicity, we adopt a 10-dim state observation consisting of the current location of the agent and the four green regions. This task requires dynamic controls/planning.

Franka Kitchen

We propose a setting (and thus a variant of the original Franka Kitchen task) where the agent is asked to complete 4 object manipulations (out of 7 different options) in an order specified by the goal. The 7 tasks are: turn on/off the bottom burner, turn on/off the top burner, turn on/off the light, open/close the slide cabinet, open/close the hinge cabinet, open/close the microwave oven, and push/move the kettle to the target location. Compared to the other variants, we use a strict criterion, i.e., the task succeeds when all 4 sub-tasks are completed, where each of them needs to be done in the requested order to be counted as completed. The environment will terminate when a sub-task other than the specified 4 is performed. The action space is based on the joint velocity (8-dim) of the robot. We use the original 30-dim state observation consisting of poses of all the relevant objects and the proprioception signals as well as an additional 14-dim goal embedding. This embedding assigns a 2-d vector to each of the 7 potential sub-tasks. Each vector is one of $[0, 0], [0, 1], [1, 0], [1, 1]$ (indicating the order to be completed for the corresponding sub-task) and $[-1, -1]$ (meaning the sub-task should not be completed). Note that we do not include the target pose of the objects in the state observation (i.e., we ask the agent to learn it from the demonstrations).

ManiSkill2

ManiSkill2 [55] is a recently proposed comprehensive benchmark for low-level object manipulation tasks. We choose 5 tasks as the testbed. (1) Stack Cube for picking up a cube, placing it on top of another, and the gripper leaving the stack. (2) Turn Faucet for turning on different faucets. (3) Peg Insertion for inserting a cuboid-shaped peg sideways into a hole in a box of different geometries and sizes. (4) Push Chair for pushing different highly articulated chair models into a specified goal location (via a mobile robot). (5) Pour for pouring liquid from a bottle into the target beaker with a specified liquid level. All tasks have all object poses fully randomized (displacement around 0.3m and 360° rotation) upon environment reset (this is in contrast to environments such as Franka Kitchen). Note that the holes in Peg Insertion have only 3mm of clearance, requiring highly precise manipulation, and it needs at least half of the peg to be pushed sideway into the holes (in contrast to the similar yet easier tasks [177]). The tasks we select involve both static and mobile manipulation and cover 3 action spaces (delta joint velocity control for Push Chair, delta end-effector pose control for Pour, and delta joint pose control for the rest). For state observations, we use states of varied dimensions across these tasks (see details in the ManiSkill2 paper) For Turn Faucet, we slightly modify the default state observation by appending an extra 3-dim vector (the pose of the faucet link) so that it is easier for the agent to distinguish between different faucet models. The corresponding demonstrations are modified as well. For point cloud observations, we use the default pre-processing strategy provided by ManiSkill2 to obtain a fixed length of 1200 points (RGB & XYZ) per timestep.

3.7.2 Details of the Demos and the Evaluation Protocol

Moving Maze

We curate demonstrations by adopting a mixture of heuristics and an RRT-style planner with hindsight knowledge not available at test time. For each randomized environment configuration, we randomly choose one of the found paths from the starting square to the target one as the demonstration trajectory. We chunk the maze into 6 regions: the three islands (each bridge belongs to the island below it; the starting square and the target square belong to the top right and the bottom left regions, respectively), each of which is further divided into two regions (cut vertically in the middle). An RRT-style sampler is used to find paths connecting adjacent regions sequentially (starting from the initial position to the second region). We restrict the number of steps in each of the paths across two adjacent regions to be ≤ 13 so that the maximum total length of a demo trajectory is $\leq 13 \times 5 = 65$ steps. To enable this type of planning with dynamic environments, we first generate demonstrations with a static version of the maze and then animate the moving elements later coherently. This is not possible during inference time as it requires hindsight knowledge. We use 400 demo trajectories for training and evaluate all agents on both these 400 configs and a held-out set of 100 unseen environment configs. We report the task success rate as the major metric. During inference, we set the maximum number of steps as 60.

Franka Kitchen

We replay a subset of the human demonstrations originally proposed in [57] in the simulator. Specifically, we randomly select 50 demo trajectories of length ranging from 150 to 300 that succeed in achieving 4 different sub-tasks out of the 7 options. We relabel them with the privileged information to construct the goal embedding described previously. Note that this embedding vector is fixed across different time steps for each individual trajectory. There are 20 total different ordered sub-task combination presented in the 50 demonstrations, where the majority of combination only has ≤ 3 trajectories. Combinatorial generalization regarding sub-tasks is too challenging in this case (there are $35 \times 4! = 840$ total combinations); so we focus on evaluating generalization w.r.t. initial robot/object poses. We use 90 unseen environment configurations (each requiring the completion of 4 sub-tasks) presented in the original human demonstrations for evaluation (we only include seen sub-task combinations). We report the task success rate (requiring the completion of all 4 sub-tasks in a trajectory) and the average number of successful sub-tasks per trajectory as the metrics. During inference, we set the maximum number of steps as 280.

ManiSkill2

For all tasks except for Push Chair, we use the original demonstrations provided by ManiSkill2, which are generated by a mixture of TAMP solvers and heuristics. Please see the original paper for details (the actual code used to generate these demonstrations is not released, though). For Stack Cube and Turn Faucet, we randomly sampled 500 demo trajectories for the training data. For Turn Faucet, we use trajectories from 10 different faucet models for the demos and perform the evaluation of 0-shot generalization on 4 unseen faucets (each with 100 different scene configurations). Not that the demonstrations of Turn Faucets have most of the faucets pushed rather than grasped, i.e., under-actuated control. For Peg Insertion, each different environment config comes with a different shape and size of the box (with the hole) and the peg. We randomly sampled 1000 trajectories from the original ManiSkill2 demonstrations as the training data and sampled 400 unseen environment configs for the 0-shot generalization evaluation. For Push Chair, we use a complicated heuristic-based approach from [114], adapted to ManiSkill2 (where it uses additional privileged information and achieves around 50% task success rate), as the demonstrator. We collected 1000 successful trajectories as the training data across 6 chair models and we evaluated the 0-shot generalization performance on 300 environment configs distributed over 3 unseen chair models. For Pour, we use 150 successfully replayed demonstration trajectories provided by ManiSkill2 to generate the point cloud observation sequences (this task does not support state observation as it involves soft-body). We use the success conditions from the original ManiSkill2 paper to report the task success rate. During inference, we set the maximum number of steps allowed as 200, 200, 200, 250, and 300 for Stack Cube, Push Chair, Peg Insertion, Turn Faucet, and Pour, respectively.

3.7.3 Details of Network Architecture and Training

Vanilla BC

: We use a Markovian policy implemented as a three-layer MLP with a hidden size of 256 and ReLU non-linearity. We train it with a constant learning rate of $1e-3$ with Adam optimizer with a batch size of 32 for 150K iterations (Moving Maze), 300K iterations (Franka Kitchen), and 500K iterations (ManiSkill2). We find training longer leads to over-fitting even with L2 regularization.

Decision Diffuser

We use the reference implementation provided by the authors of DD and make the following changes in the diffusion model: 100 diffusion steps, 20 context size, and 4 horizon length (in our experiments we found that longer performs worse). The diffusion and inverse-dynamics models have $\sim$1.6M parameters in total. Since DD works on fixed sequence lengths, we pad the start and end states during training and only the start states during inference.

Decision Transformer

We adopt the minGPT implementation and use the same set of hyperparameters for all tasks (a feature embedding size of 128 and 4 masked multi-head attention layers, each with 8 attention heads), totaling slightly greater than 1M learnable parameters. The action decoder is a 3-layer MLP of two hidden layers of size 256 with ReLU non-linearity (except that for Franka Kitchen we use a 2-layer MLP of hidden size 1024). We train DT with a learning rate of $5e-4$ with a short warm-up period and cosine decay schedule to $5e-5$ for all tasks (except for Franka Kitchen, whose terminal lr is $5e-6$) with the Adam optimizer with a batch size of 256. We train for 200K iterations for Moving Maze and Franka Kitchen and train for 500K iterations for all tasks in ManiSkill2. We use a weight decay of 0.0001 for all tasks but Franka Kitchen (for which we use 0.1). We use a context size of 10 for ManiSkill2 tasks, 20 for Moving Maze, and 10 for Franka Kitchen. We use learnable positional embedding for the state and action tokens following the DT paper.

Behavior Transformer

We started with the configuration used for the Franka Kitchen task in the original paper. We changed the number of bins in K-Means to 1024 (we find that for our tasks, a smaller number of bins works worse) and changed the context size to 10 (in line with the other transformer-based models). The Transformer backbone has approximately the same number of parameters ($\sim$1M) as CoTPC and DT. We train the model for around 50k iterations (we find that training longer leads to over-fitting easily for BeT, potentially because of its discretization strategy and the limited demos used for BC). For ManiSkill2 tasks, we use the same architecture as that for DT, except that we use the center plus offset predictors to decode the actions. We train 100K iterations for Turn Faucet, Push Chair, and Pour. We train 200K iterations for Peg Insertion and Stack Cube.

CoTPC

Unless specified here, we keep other configurations (both model training and network architecture) the same as those in BeT. We use no positional embeddings for CoT tokens as they themselves are learnable prompts. The CoT decoder is a 2-layer MLP with ReLU non-linearity of hidden size 256. We use a coefficient $\lambda = 0.1$ for the auxiliary MSE loss for all tasks. During training, we apply random masking to the action and state tokens so that the CoT tokens attend to a history of varied length (from the first step to a randomized t-th step). Other details have already been presented previously.

3.7.4 Details of Point Cloud-based CoTPC

To process point cloud observations, we adopt a lightweight PointNet [123] ($\sim$27k parameters) that is trained from scratch along with the transformer in an end-to-end manner. We concatenate additional proprioception signals with the point cloud features to the input state tokens. In CoTPC where the CoT decoder is trained to predict the

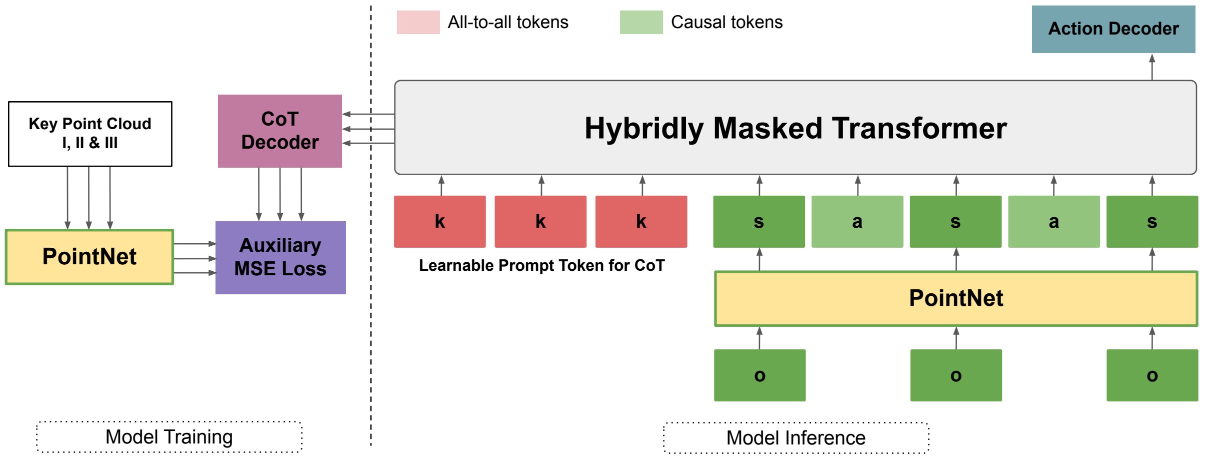


Figure 3.7. Illustration of the point cloud-based CoTPC. Compared to state-based CoTPC, we add a PointNet to process the point cloud observations as well as the point cloud CoT. We omit the data path for the input proprioception signals to the model.

point cloud CoT, we ask the decoder to predict PointNet features of the CoT instead. We find that the auxiliary point cloud CoT loss causes the PointNet encoded representations to collapse. Inspired by [26], we use a stop-gradient operation in the point cloud CoT encoding path to prevent this. We illustrate the network architecture in Fig. 3.7. The training strategies (we use the same set of hyperparameters) and evaluation protocols are similar to those of the state-based experiments.

3.8 Conclusion and Discussion

In this work, we propose CoTPC, a hierarchical imitation learning algorithm for learning generalizable policies from scalable but sub-optimal demos. We formulate the hierarchical principles (in the form of chain-of-thought) in offline policy learning with a novel Transformer-based design and provide an effective way to obtain chain-of-thought supervision from demonstrations in an unsupervised manner. Our approach empirically validates the benefits of task decomposition even for short-horizon tasks. We demonstrate that CoTPC can solve a wide range of challenging low-level control tasks, consistently outperforming many existing methods.

Many existing work dealing with "long-horizon" robotic tasks (SayCan [1], ALFRED [149], etc.) assumes that low-level control is solved or that the task hierarchy is given. On the contrary, in this paper, we study better ways to learn to solve low-level control tasks with unsupervisedly discovered hierarchical information as supervisions. We believe that CoTPC can be extended in a multi-task learning setup, i.e., policy learning from diverse demos across different low-level control tasks.

Acknowledgement

Chapter 3, in full, is a reprint of the manuscript: "Chain-of-Thought Predictive Control" (Zhiwei Jia, Vineet Thumuluri, Fangchen Liu, Linghao Chen, Zhiao Huang, Hao Su), whose preliminary version has been presented at the Workshop on Reincarnating Reinforcement Learning at ICLR 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Task Decomposition by Partitioning the Task Space

While most Embodied AI tasks can be decomposed temporarily into sub-tasks or subskills, as presented in the previous two chapters, task decomposition can also refer to partitioning the task space itself. In this chapter, we introduce how to tackle the challenges of large-scale policy optimization over diverse environment variations, which arise from acquiring generalizable deep RL agents that can potentially work in unseen environment variations. One of the key ideas here is to decompose the space of task variations and learn to solve the task in a divide-and-conquer manner. Specifically, we observe that an agent (a generalist) trained on many variations tends to learn faster at the beginning yet plateaus at a less optimal level for a long time. In contrast, an agent (a specialist) trained only on a few variations can often achieve high returns under a limited computational budget. We, therefore, propose an RL framework to tackle complex tasks in a distributed training manner to have the best worlds of both worlds. Several key ablations reveal when and how to launch the specialist agents as well as how to merge them back into a single generalist agent. Our proposal is a meta-framework[1] that brings performance boost to various baseline online RL methods over several popular RL benchmarks.

[1]As a meta-algorithm, the (pseudo)code is available here.

4.1 Introduction

Deep Reinforcement Learning (DRL) holds promise in a wide range of applications such as autonomous vehicles [43], gaming [150], robotics [81] and healthcare [180]. To fulfill the potential of RL, we need algorithms and models capable of adapting and generalizing to unseen (but similar) environment variations during their deployment. Recently, several benchmarks [56, 31, 107, 159, 50, 190] were proposed to this end, featuring a very high diversity of variations in training environments, accomplished through procedural generation and layout randomization, to encourage policy generalization.

However, due to the sheer number of variations, many existing DRL algorithms struggle to efficiently achieve high performance during training, let alone generalization. For instance, in Procgen Benchmark [31], a PPO [144] agent trained on a thousand levels can have poor performance even with hundreds of millions of samples. Training PPO on visual navigation tasks involving $\sim$100 scenes might require billions of samples to achieve good performance [172]. Several lines of work have been proposed to alleviate this issue, by accelerating training with automatic curriculum [80], improving learned representations with the help of extra constraints [68, 129], or decoupling the learning of the policy and value networks [34, 128].

Orthogonal to these approaches, we tackle the challenge from a perspective inspired by how human organizations solve difficult problems. We first define a *generalist* agent to be a single policy that can solve all environment variations. We also define a *specialist* agent to be a policy that masters a subset, but not all, of environment variations. Our goal is to utilize experiences from the *specialists* to aid the policy optimization of the *generalist*.

We observe that trajectories belonging to different environment variations often consist of shared early stages and context-specific later stages. It is often more efficient for a single generalist to learn the *shared* early stages for all contexts than first training different specialists and then merging them into a generalist. For instance, learning to push a chair towards a goal, regardless of variations in chair geometry, topology, or dynamics, requires an agent to first recognize the chair and approach it. During these early stages, jointly training an agent on all chairs, starting and goal states results in faster learning. However, as the visited states get more and more diverse, it becomes increasingly hard for the policy and value network to maintain the predictive power without forgetting (i.e., “catastrophic forgetting”), or be vigilant to input dimensions that were not useful at the beginning but crucial for later stages (i.e., “catastrophic ignorance” in Sec. 4.4). This poses a significant challenge and results in performance plateaus. Meanwhile, if we only consider a small subset of environment variations and train a specialist on it, then due to the low state variance, the specialist can often master these variations, achieving a higher return on them than the generalist.

Inspired by these observations, we propose a novel framework, named **Generalist -Specialist Learning (GSL)**, to take advantage of both *specialist*'s high return and *generalist*'s faster early training. As illustrated in Fig. 4.1, we first optimize the generalist on all environment variations for fast initial policy learning. When it fails to improve (by simple criteria), we launch a large population of specialist agents, each loaded from the generalist checkpoint, and optimize on a small subset of environment variations. After specialists' performance quickly surpasses the generalist's, we use specialists' demonstrations as auxiliary rewards for generalist training, advancing its own performance. While some previous approaches also involve specialist training [107, 163, 52, 106, 25], they either do not realize or diagnose the gains and losses for generalist vs. specialist, or focus on a different setup with their essential idea orthogonal to our proposal. We demonstrate the effectiveness of our framework on subsets of two very challenging benchmarks: Procgen [32] that consists of procedurally generated 2D games (with 1024 training levels, more than the official 200 levels setup), and ManiSkill [107] that evaluates physical manipulation skills over diverse 3D objects with high geometry and topology variations.

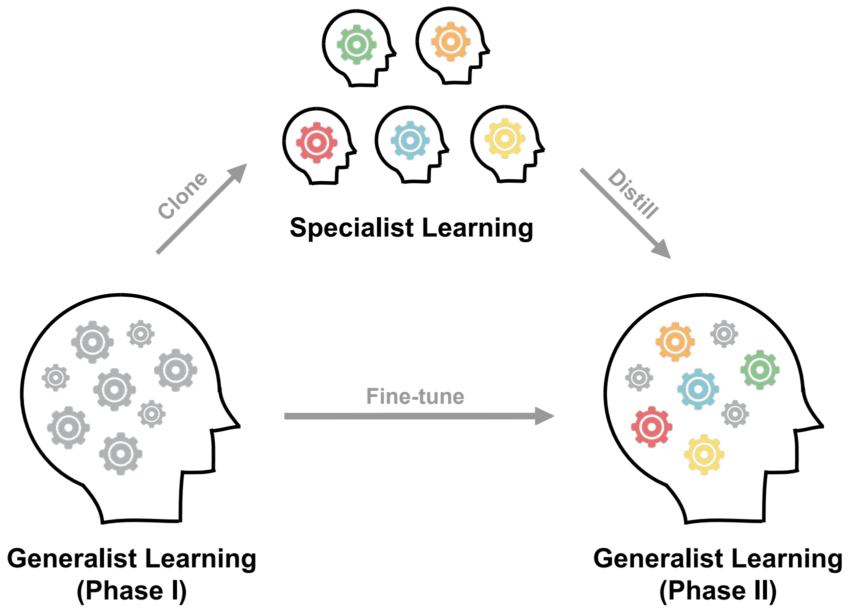


Figure 4.1. Illustration of Generalist-Specialist Learning (GSL) framework. During initial generalist learning (phase I), a generalist agent learns to master all environment variations at once. Next, each specialist agent works on a subset of environment variations. Finally, the generalist is fine-tuned (phase II) with guidance provided by the specialists (e.g., via demonstrations).

4.2 Related Work

Divide-and-Conquer in RL

Our work is most closely related to works along this line. Previous works like [163, 52] have adopted divide-and-conquer for training an RL agent. They split the state space into subsets, and alternate between training each local policy on each subset and merging the local policies into a single center policy using imitation learning. Although these approaches also adopt the perspective of generalists and specialists, they did not study the timing for starting specialist training, which is a key contribution of our work. In addition, when distilling experiences from specialists into a generalist, they usually use behavior cloning and address the demonstration inconsistency issue by KL divergence regularization between local policies [52], which requires a synchronized training strategy, making it intractable when scaling to large numbers of specialists as in our experiments.

We show that our approach can still work well with a large number of specialists.

Policy Distillation and Imitation Learning

Distilling a single generalist (student) from a group of specialists (teachers) is a promising way to achieve good performance on challenging tasks [138, 137, 106, 36, 162]. Population-based agent training (PBT) was utilized in [36, 162]. Previous works like [138] have adopted supervised learning to train a generalist over specialists' demonstrations. Other works convert demonstrations into rewards for online learning [44, 63, 183, 132, 147]. Our framework makes good use of the policy distillation as its sub-module.

Large-Scale RL

Training an RL agent over a large number of environment variations is a promising approach to obtaining a generalizable policy [32, 107]. One series of benchmarks for this objective involves variations of object geometry and topology [107] and task semantics [182, 71], which are usually mentioned as multi-task RL benchmarks. Another series of benchmarks procedurally generate diverse levels and layouts for an environment [166, 32]. For both series of benchmarks, training a single agent over multiple variations is known to be challenging, which warrants more exploration into this field.

Multi-task RL

In multi-task RL, an agent is trained on multiple tasks given a task-specific encoding. Recent works have made significant progress in accelerating policy optimization across multiple tasks. One stream of work focuses on improving task (context) encoding representations from environment dynamics or reward signals [9, 154]. Another stream focuses on studying and alleviating negative gradient interference from different tasks [141, 181, 92]. Different from these approaches, our framework is designed for general RL tasks, which not only encompasses multi-task RL environments but also general RL environments such as Procgen, whose environment variations do not have semantic encoding (i.e., each variation is represented by a random seed).

4.3 Background

A general **Markov decision Process** (MDP) is defined as a tuple

$$M = (S, A, T, R, \gamma)$$

where S, A are state space and action space, $T(s'|s, a)$ is the state transition probability, $R(s, a)$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. In reinforcement learning, we aim to train a policy $\pi(a|s)$ that maximizes the expected accumulated return given by $J(\pi) = E_{(s_t,a_t)\sim\rho_\pi}[\sum_{t=0} \gamma^t r(s_t, a_t)]$.

A **Block Contextual Markov Decision Process** (BC-MDP) [186, 40, 155] augments an MDP with context information, which can be defined as $(C, A, M(c), \gamma)$, where C is the context space, M is a function which maps any context $c \in C$ to an MDP $M(c) = \{S^c, T^c, R^c\}$. BC-MDP can be adapted to the multi-task setting, where contexts control objects used in the environments (e.g. object variations in ManiSkill [107]) and task semantics (e.g. tasks in MetaWorld [182]). BC-MDP can also be adapted to a general MDP to control the random seeds (e.g. seeds for procedural generation in Procgen [32]). In this paper, we consider both settings. Unlike previous works [155], our framework does not require the context to contain any semantic meaning, which is typical in the multi-task RL setting. The context can be anything that splits an MDP into several sub-MDPs.

In the following sections, we first motivate our approach by analyzing a simple example in Section 4.4. We then introduce several important ingredients in our Generalist-Specialist Learning (GSL) framework.

4.4 An Illustrative Example

Consider a simple "brush"-like maze illustrated in Fig. 4.3a. An agent (a *generalist*) starts from the leftmost position in the corridor and needs to reach the goal specified by a

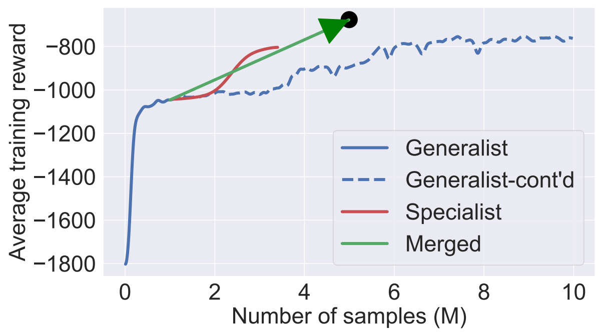
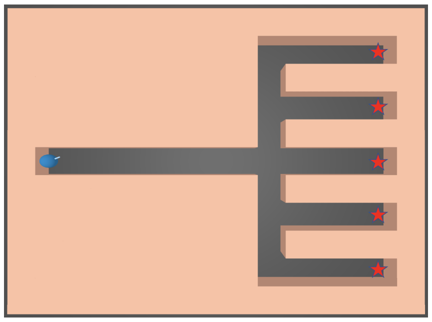


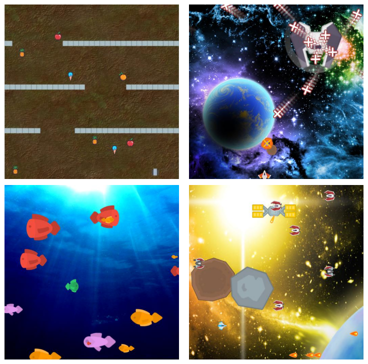
Figure 4.2. Training curve of PPO on the brush-like maze. The generalist learns fast but plateaus quickly (dashed blue line). The specialists (cloned from the generalist) learn to solve individual goals better (solid red line). The fine-tuned generalist (trained by DAPG) with the specialists' demos achieves the best results efficiently (green arrow). This improvement is consistent among 5 runs.

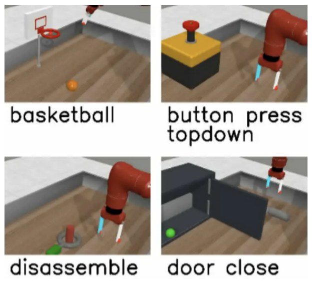
real context c. For each $i = \{1, 2, \cdots, 5\}$, $C_i = (\frac{i-1}{5}, \frac{i}{5}]$ indicates the context space for the i-th goal (marked as red stars). Upon environment resets, we first uniformly sample the goal i and then sample c uniformly from C_i. An agent is given the position and velocity (both are real vectors) as an observation and is capable of setting its velocity as an action. The reward is the approximated negative geodesic distance between the current position and the goal.

We train a PPO agent and discover the phenomenon of "*catastrophic ignorance*" (we coin this term inspired by "catastrophic forgetting"). As in Fig. 4.2, in the early stages, the agent quickly learns to move rightwards in the maze. However, the agent tends to ignore the goal context c since it plays little role in the early stages. This is revealed by the agent's tendency to always move rightwards after the intersection and arrive at the middle goal regardless of the goal context. It requires a large amount of samples for the agent to learn to extract features about the goal context.

To overcome this challenge, at the performance plateau of the *generalist*, we split the environment into 5 environment variations, each with a different goal (context interval), and initiate 5 *specialists* from the generalist checkpoint to master each of the variations. For each specialist, because the reward distribution is identical across environment resets,


(a) Illustrative environment


(b) Procgen



(c) Meta-World

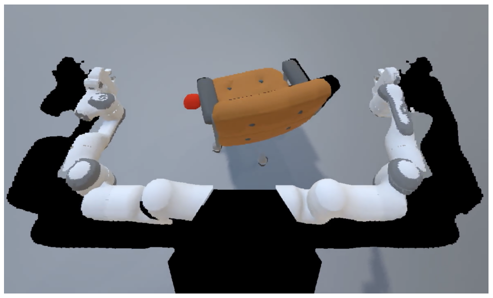

(d) ManiSkill

Figure 4.3. (a) An illustrative environment that consists of a long corridor and a "brush"-like maze, where the agent starts from the left and needs to reach one of the five targets (stars in the figure) specified by a goal context. (b) Sample environments in Procgen, a 2D vision-based game environment where levels are generated procedurally. (c) Sample environments in Meta-World, where the agent is asked to perform diverse manipulation tasks given state-space observations. (d) The PushChair task in ManiSkill, where an agent is required to push a chair towards a goal (red ball) given point cloud observations of diverse chairs.

the specialist can quickly succeed even though it does not understand the semantic meaning of the context. Then, we use specialists to generate demonstrations. In practice, auxiliary rewards can be generated from the demos using techniques such as Demonstration Augmented Policy Gradient (DAPG, [132]) and Generative Adversarial Imitation Learning (GAIL, [63]). With strong rewards obtained from demonstrations, the generalist is not bothered by the challenges in path-finding and can focus on discriminating goal context. As a result, it quickly learns (as shown in Fig. 4.2) to factor the previously ignored goal context into making its decisions, thereby overcoming the performance plateau.

Besides that catastrophic ignorance can be cured by our framework, GSL can also deal with the catastrophic forgetting issue. As the generalist learning proceeds, it becomes increasingly hard for its policy/value network to maintain the predictive power for all the environment variations without forgetting. The introduction of specialists can mitigate this issue because each specialist usually only needs to work well on a smaller subset of the environment variations and this specialized knowledge is transferred and consolidated into a single agent with the help of the collected demonstrations. As catastrophic forgetting is well-known to the neural network community, we do not provide an illustrative example here.

Algorithm 1. GSL: Generalist-Specialist Learning

```
Require: (1) Environment $E$ with context space $C$ (2) Number of specialists $N_s$ (3)
    Number of env. variations for specialist $N_{lenv}$ (4) Number of demonstrations $N_D^g$ from
    generalist and $N_D^s$ from specialists (5) Performance plateau criteria $\mathcal{H}$
 1: Initialize generalist policy $\pi^g$
 2: Train $\pi^g$ on $E$ until $\mathcal{H} = 1$                                ▷ e.g., PPO, SAC
 3: if $\pi^g$ optimal enough then
 4:     Exit                                                        ▷ done with GSL
 5: end if
 6: Find the $N_{lenv}$ lowest-performing environment variations from $E$, collectively denoted
    as $E_{low}$.
 7: Split $E_{low}$ into $N_s$ disjoint environment variations $\{E_i\}$ by splitting the context space
    $C$.
 8: Obtain $\pi^g_{low}$ by fine-tuning $\pi^g$ on $E_{low}$                           ▷ optional
 9: for each $i = 1 \cdots N_s$ do                                          ▷ in parallel
10:     Initialize specialist $\pi_i^s = \pi^g$ or $\pi^g_{low}$
11:     Train $\pi_i^s$ on $E_i$
12:     Generate $\frac{N_D^s}{N_s}$ demos $\mathcal{T}_i$ with $\pi_i^s$ on $E_i$
13: end for each
14: Generate $N_D^g$ demos $\mathcal{T}_g$ with $\pi^g$ on $E \backslash E_{low}$
15: Continue training $\pi_g$ on $E$ with auxiliary rewards induced from $\{\mathcal{T}_i^D\} \cup \mathcal{T}_g$ (via DAPG,
    GAIL, etc.)
```

4.5 Generalist-Specialist Learning

Motivated by our previous example, we now introduce our GSL framework. At a high level, the framework is a “meta-algorithm” that integrates a reinforcement learning algorithm and a learning-from-demonstration algorithm as building blocks and produces a more powerful reinforcement learning algorithm. While there exists works with a similar spirit, we identify several design choices that are crucial to the success but were not revealed in the literature. We will first describe the basic framework, and then introduce our solutions to the key design choices that lead to improved sample complexity in environments that are too difficult for the building block reinforcement learning algorithm due to the catastrophic forgetting and ignorance issues.

4.5.1 The Meta-Algorithm Framework

We first initialize a generalist policy π^g and train the model over all variations of the environment using actor-critic algorithms such as PPO [145] and SAC [58]. When the performance plateau criteria $\mathcal{H}$ (introduced later) is satisfied, we stop the training of π^g. This could occur either when π^g reaches optimal performance (in which case we are done), or when the performance is still sub-optimal. If the performance is sub-optimal, we then split all environment variations into small subsets, and launch a population of specialists, each initiated from the checkpoint of generalist, to master each subset of variations. We finally obtain demonstrations from the specialists and resume generalist training with auxiliary rewards created by these demonstrations. The basic framework is outlined in Algorithm 1.

4.5.2 When and How to Train Specialists

While there exist attempts to use the divide-and-conquer strategy to solve tasks in diverse environments, a systematic study of the timing to start specialist training is missing. The default choice in the literature [163, 52] starts specialist training from the very beginning and periodically distills the specialists into the generalist. However, as in Fig. 4.8, we observe that training the specialists before the generalist's performance plateaus does not take full advantage of the generalist's fast learning during the early stages, and therefore results in less optimal sample complexity. Consequently, we start specialist training only after the generalist's performance plateaus.

Performance plateau criteria for generalists. We introduce a binary criteria $\mathcal{H}$ to decide when the performance of generalist plateaus. In our implementation, we design simple but effective criteria based on the change in average return, which works well in our benchmarks. Given returns from M epochs $\{R_1, \ldots, R_M\}$, we first apply a 1D Gaussian filter with kernel size 400 to smooth the data. Then

$$\mathcal{H}(t) = \mathbb{1}(R_t + \epsilon \geq R_{t'},\ \forall\ t' \in\ \{t+1, t+2..., t+W\})$$

Intuitively, the criteria are satisfied if the smoothed return at a certain epoch is approximately higher than (more than a margin ϵ) all smoothed returns in the future W epochs.

Assigning environment variations to specialists. When we assign sets of environment variations to specialists, we hope that each specialist can master their assigned variations, yet we also hope that the number of specialists N_s is not too large if the number of all training environment variations is already large (in which case we need a large amount computational resource to train the specialists in parallel). Empirically, we observe that the generalist can solve some environment variations reasonably well, yet performs poorly on others. Therefore, we launch specialist training only on the N_{lenv} lowest performing environment variations.

We empirically find that an optional step (line 8 of Alg. 1), specifically fine-tuning π^g on the N_{lenv} lowest performing environment variations before training the specialists, can help to improve sample efficiency as it gives the specialists a better starting point. Specifically, we fine-tune π^g for 200 epochs, or $200 * 16384$ samples (a very small number compared to the 100M total budget) and find this step helpful for tasks in Procgen if PPO is used as the backbone RL algorithm.

After we assign the variations to specialists, we start to train these specialists in parallel. We assume that a specialist can always solve the environment with a few variations. For example, for most tasks in Procgen which contain 1024 procedurally generated levels for training, we find $N_{lenv} = 300$ and $N_s = 75$ good enough (i.e. each specialist is trained on $300/75 = 4$ variations). Therefore, we can train the specialists until they accomplish their assigned variations. In practice, we set a fixed number of samples N_{sample} for training each specialist.

4.5.3 Generalist Training Guided by Specialist Demos

After training each specialist to master a small set of environment variations, we still need the common generalist to consolidate specialist experiences and master *all* training environment variations. In our proposed framework, we first collect the demonstration set $\{\mathcal{T}_i^D\}$ using the specialists on their respective training environment variations (we only collect trajectories whose rewards are greater than a threshold τ). Specifically, we use the best-performing model checkpoint stored by each specialist to generate the demonstration set. To ensure training stability, we also collect demos $\mathcal{T}_g$ for the remaining training variations using the generalist. We then resume generalist training using a *learning-from-demonstrations algorithm* by combining the environment reward and the auxiliary rewards induced from these demonstrations. To train a generalist in this process, we can adopt many approaches, such as Behavior Cloning (BC), Demonstration-Augmented Policy Gradient (DAPG, [132]), and Generative-Adversarial Imitation Learning (GAIL, [63]).

It is a key design factor to choose this learning-from-demonstrations algorithm. The crucial challenge comes from the inconsistency of specialist behaviors in similar states. To our knowledge, previous work of divide-and-conquer RL uses BC to distill from specialists in an offline manner; however, even if different environment variations share the same reward structure and various regularization techniques can be added to the specialist training process, we find it not scalable to the number of specialists and it fails to work well in diverse environments such as Procgen or ManiSkill in our paper. Moreover, pure offline methods such as BC usually achieve inferior performance since they are limited to a fixed set of demonstrations.

With the demos collected from specialists, we find online learning from demonstration methods such as DAPG and GAIL to be quite effective. For DAPG and GAIL, besides utilizing all the collected demos $\{\mathcal{T}_i^D\} \cup \mathcal{T}_g$, we also let the generalist interact with the environment to obtain online samples. From here, we use ρ_D and ρ_π to denote a batch of transitions sampled from the demonstrations and from the environment, respectively. While DAPG and GAIL in principle can be adapted to any RL algorithms (PPO, PPG, SAC, etc.), we evaluate GSL on challenging benchmarks using their corresponding strong baseline RL algorithms (PPO/PPG on Procgen, PPO on Meta-World, and SAC on ManiSkill). Below, we derive a formula to illustrate how we adapt DAPG and GAIL in our experiments.

We modify DAPG for DAPG + PPO. We first calculate the advantage value $A(s, a)$ for $(s, a) \sim \rho_\pi$ using GAE [143]. Then, in each PPO epoch, we compute the maximum advantage denoted as $\hat{A}$. We obtain the overall policy loss (value loss omitted here):

$$\mathcal{L}_\rho^{CLIP}(\theta) = -\mathbb{E}_{(s,a)\sim\rho}$$

$$\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A(s,a), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A(s,a)\right)\right]$$

$$\mathcal{L}_\rho^1(\theta) = -\mathbb{E}_{(s,a)\sim\rho}[\pi_\theta(a|s)]$$

$$\mathcal{L}_{DAPG+PPO}(\theta) = \mathcal{L}_{\rho_\pi}^{CLIP}(\theta) + \omega \cdot \hat{A} \cdot \mathcal{L}_{\rho_D}^1(\theta)$$

We find that a smoothed loss $\mathcal{L}^1(\cdot)$ here significantly improves training stability for some tasks, compared to the cross entropy (style) loss used in the original DAPG paper.

We set $\omega = 0.5$ in all our experiments and find that decreasing it over time (as in the original DAPG paper) can lead to worse performance.

For GAIL + SAC, we train a discriminator to determine whether a transition comes from policy or from demonstration. We obtain the following losses for policy π_ϕ, discriminator D_ψ, and Q-function Q_θ:

$$\mathcal{L}_{GAIL+SAC} = \mathcal{L}_\pi(\phi) + \mathcal{L}_D(\psi) + \mathcal{L}_Q(\theta)$$

$$\mathcal{L}_\pi(\phi) = -\mathbb{E}_{s_t \sim \rho_\pi}\left[\mathbb{E}_{a_t \sim \pi_\phi}\left[\alpha \log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t,a_t)\right]\right]$$

$$\mathcal{L}_D(\psi) = \mathbb{E}_{\rho_\pi}[\log(D_\psi(s_t,a_t))] + \mathbb{E}_{\rho_D}[\log(1-D_\psi(s_t,a_t))]$$

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{\rho_\pi \cup \rho_D}[(Q_\theta(s_t,a_t) - (\tilde{r}(s_t,a_t) + \gamma V_{\bar{\theta}}(s_{t+1})))^2]$$

$$\tilde{r}(s_t, a_t) = \beta r(s_t, a_t) + (1-\beta)\log(D_\psi(s_t, a_t))$$

here α is the temperature in SAC; β is the hyper-parameter that interpolates between the environment reward and the reward from the discriminator; moreover, $V_{\bar{\theta}}(s_t) = \mathbb{E}_{a_t \sim \pi_\phi(\cdot|s_t)}\left[Q_\theta(s_t, a_t) - \alpha \log(\pi_\phi(a_t|s_t))\right]$ is target value.

PPG [34] is a recently proposed RL algorithm that equips PPO with an auxiliary phase for learning the value function. It achieves state-of-the-art training performance on Procgen. We therefore also evaluate the PPG + DAPG combination on Procgen, with essentially the same adaptation as for PPO + DAPG. Our experiments illustrate the generality of our framework as a meta-algorithm.

4.6 Experiments

We evaluate our Generalist-Specialist Learning (GSL) framework on three challenging benchmarks: Procgen [32], Meta-World [182] and SAPIEN Manipulation Skill Benchmark (ManiSkill Benchmark [107]). To begin with, we introduce the benchmark environments (the tasks) used in our evaluation.

4.6.1 Environments

Procgen

Procgen is a set of 16 vision-based game environments, where each environment leverages seed-based procedural generation to generate highly diverse levels. All Procgen environments use 15-dimensional discrete action space and produce $(64, 64, 3)$ RGB observation space. In our experiments, we use 1024 levels for training, which is different from the original 200-training-level setup (as we try to evaluate how well GSL scales). We select the 7 most challenging environments under our setting based on the normalized score obtained by training the baseline PPO algorithm (PPO can achieve a close to perfect score on the remaining environments given 100M total samples). These environments are BigFish, BossFight, Chaser, Dodgeball, FruitBot, Plunder, and StarPilot. A subset of environments is illustrated in Fig. 4.3b. This benchmark leverages procedural generation and is also suitable for evaluating the generalization performance of our framework. We adopt the IMPALA CNN model [41] as the network backbone.

Meta-World

Meta-World is a large-scale manipulation benchmark for meta RL and multi-task RL featuring 50 distinct object manipulation skills (see samples in Fig. 4.3c). We choose multi-task RL as our testbed, including MT-10 and MT-50 (with 10 and 50 skills to learn, respectively), which only evaluate the agents' performance in the training environments. The state space for this benchmark is high-dimensional and continuous, representing coordinates & orientations of target objects and parameters for robot arms as well as encodings for the skill ID. We use the V2 version of the benchmark.

ManiSkill

ManiSkill is a recently proposed benchmark suite designated for learning generalizable low-level physical manipulation skills on 3D objects. The diverse topological and geometric variations of objects, along with randomized positions and physical parameters, lead to challenging policy optimization. Since realistic physical simulation and point cloud rendering processes (see Fig. 4.3d) are very expensive for ManiSkill environments (empirically we observe that the speed is at 30 environment steps per second), we only evaluate our framework on the PushChair task. In this task, an agent needs to move a chair towards a goal through dual-arm coordination (see Fig. 4.3d). The environment has a 22-dimensional continuous joint action space. Each observation consists of a panoramic 3D point cloud captured from robot cameras and a 68-dimensional robot state (which includes proprioceptive information such as robot position and end-effector pose). In our experiments, we use a smaller scale of the PushChair environment that consists of 8 different chairs, where we already find our framework significantly improves the baseline. We also transform all world-based coordinates in the observation space into robot-centric coordinates. Different from the original environment setup, we add an indicator of whether the robot joints experience force feedback due to contact with objects, and we do not reset the environment until the time limit is reached. We adopt the PointNet + Transformer over object segmentations model in the original baseline as our network backbone.

4.6.2 Main Results

We first train a baseline PPO model on Procgen and MT-10 and MT-50 from Meta-World, along with a baseline SAC model on ManiSkill. We then compare these baselines with our GSL framework where PPO+DAPG is trained on Procgen and Meta-World and SAC+GAIL is trained on ManiSkill. We demonstrate the results in Fig. 4.4 & 4.5 and Tab. 4.1 & 4.2. For better display, we only demonstrate training curves for one run per environment/task. We present additional results for multiple seeds in the next section. We also perform experiments with PPG+DAPG on the first two tasks of Procgen to further verify the generality of GSL (See Tab. 4.1). Since our GSL framework involves online interactions from specialists, *we perform necessary scaling to reflect the actual total sample*

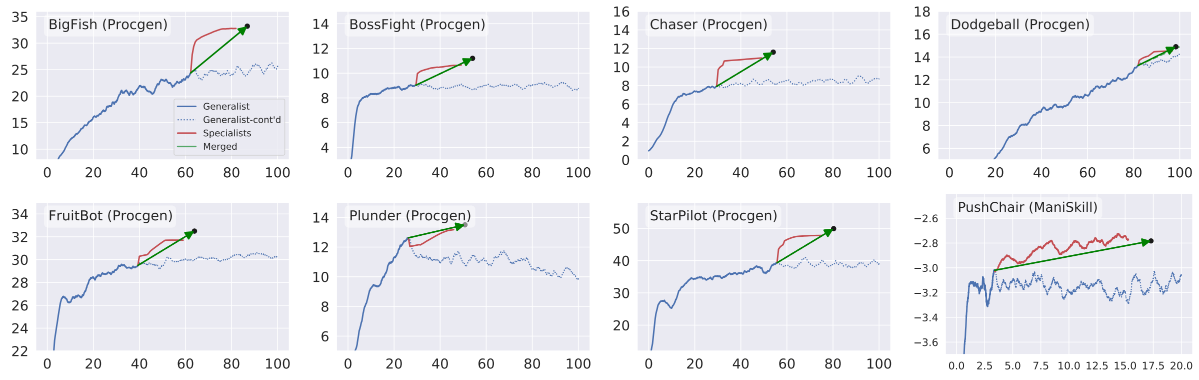


Figure 4.4. GSL significantly improves baselines (in blue) on Procgen (PPO) and ManiSkill (SAC) for their most challenging environments, respectively. The X-axis is the overall training samples (M), and the y-axis is the return. The unit for the y-axis in PushChair is 1000. The dashed blue line indicates what happens if we choose to continuously train the generalist. For clarity, we only show the starting and ending points for the step of converting specialist experiences into the generalist (green arrow) and only plot one run for better display (see Fig. 4.6 for aggregated training curves across multiple runs). We report the numerical results in Tab. 4.1.

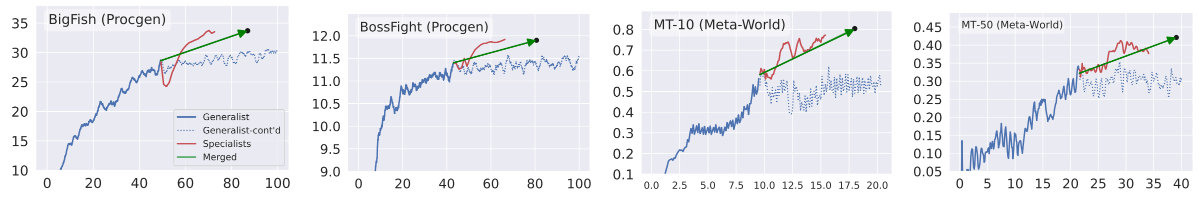


Figure 4.5. We also evaluate GSL on MT-10 & MT-50 (PPO) and on the first two challenging tasks of Procgen (with PPG as the baseline, which already achieves close-to-optimal performance). Again, GSL consistently improves the baselines. The unit for the x-axis is a million steps. Similar to Fig. 4.4, we only plot one run out of 5 for clarity. See Tab. 4.1 & 4.2 for numerical results.

complexity of our framework.

We observe that the generalist learns fast at the beginning, yet its performance plateaus at a sub-optimal level. However, as soon as we launch specialist training, they quickly master their assigned variations. The strong rewards obtained from specialists' demonstrations efficiently and effectively lift the generalist out of performance plateaus, resulting in significant improvement over baselines. These observations also corroborate those in Fig. 4.2 of our illustrative example.

We compare the final average performance of GSL with PPO/PPG baselines on (1)

Table 4.1. Our GSL framework outperforms PPO and PPG baselines on both training and generalization. Models are trained over 1024 levels for each environment and tested over 1000 unseen levels. Results are averaged over 5 runs with std. from raw episode rewards.

	BigFish	BossFight	Chaser	Dodgeball	FruitBot	Plunder	StarPilot
PPO-Train	24.6±0.7	8.6±0.2	8.5±0.3	13.7±0.3	30.1±0.6	10.5±0.8	39.4±1.4
GSL-Train	**31.1±0.8**	**11.3±0.2**	**11.5±0.3**	**15.5±0.2**	**31.9±0.3**	**13.4±0.4**	**49.5±0.4**
PPO-Test	24.3±1.1	8.6±0.3	7.9±0.4	12.7±0.3	29.1±0.5	9.7±0.5	38.0±0.9
GSL-Test	**30.0 ±0.5**	**10.4 ±0.2**	**10.9±0.2**	**14.1±0.3**	**30.5±0.4**	**13.1±0.3**	**48.7±0.5**

	BigFish	BossFight
PPG-Train	29.4±1.1	11.3±0.2
GSL-Train	**33.5±1.3**	**11.9±0.2**
PPG-Test	28.0±0.9	11.1±0.2
GSL-Test	**30.9 ±0.8**	**11.6±0.2**

Table 4.2. GSL boosts training efficiency on MT-10/MT-50 and PushChair. Results are averaged over 5 and 3 runs, respectively.

	MT-10 (%)	MT-50 (%)
PPO-Train	58.4±10.1	31.1±4.5
GSL-Train	**77.5±2.9**	**43.5±2.2**

	PushChair (k)
SAC-Train	-2.97±2.7
GSL-Train	**-2.78±2.3**

the 1024 training levels and (2) the 1000 hold-out test levels of Procgen. We observe that our framework not only improves over the baseline on the training environment variations but also on unseen environment variations, demonstrating our framework's effectiveness for obtaining generalizable policies.

4.6.3 Additional Results

In this section, we present the training curves aggregated across multiple runs. For experiments on Procgen and Meta-World, we perform 5 runs for both the baseline and GSL in each environment; for ManiSkill, we perform 3 runs (due to its high computation complexity). We illustrate the mean rewards and their standard deviations for each aggregated training curve. Notice that, due to the performance plateau criteria $\mathcal{H}$, each run has different starting and ending points for both the generalists and the specialists. We therefore normalize the x-axis to align each training curve. Specifically, we use percentage

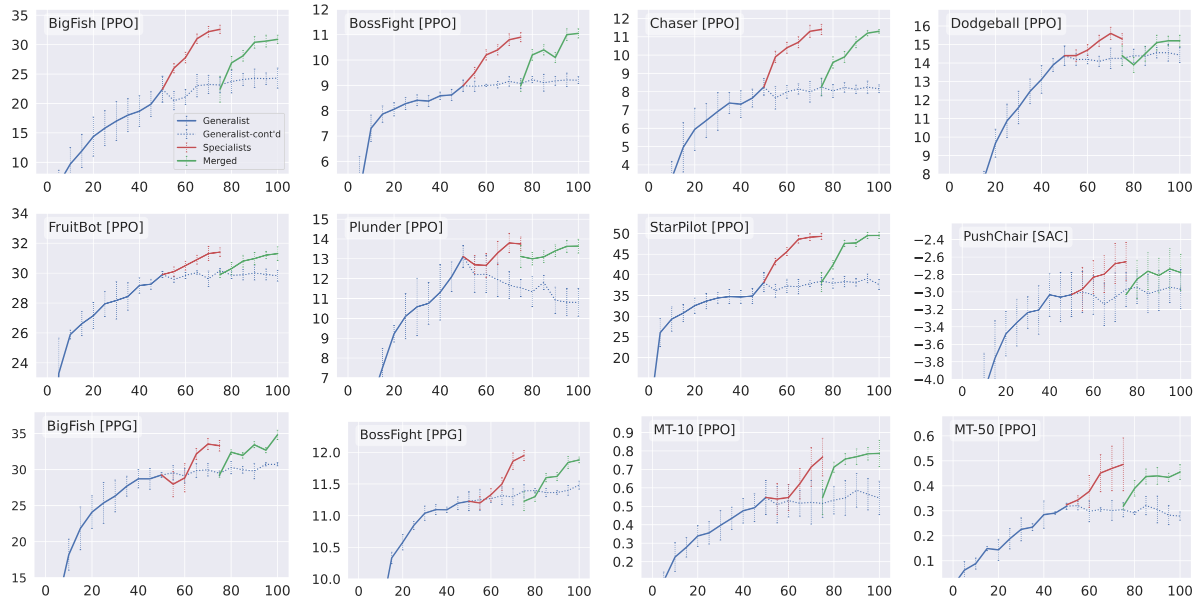


Figure 4.6. Aggregated training curves for GSL across multiple runs in Procgen, Meta-World and ManiSkill, where GSL consistently improves the baseline (dashed blue curves). The y-axis units are raw episode rewards for Procgen, average success rate for Meta-World and 1000 for ManiSkill. To align the training curves across different runs, we use percentage to represent the x-axis (0 ∼ 50% for initial generalist training, 50 ∼ 75% for specialists, and 75 ∼ 100% for specialist-guided generalist training).

(ranging from 0 to 100) as the x-axis. We display curves for the initial generalist training in the first 50%, those for the specialists in the next 25%, and finally the fine-tuned generalist in the remaining 25%.

Since the training curves plotted here are subject to smoothing, their values (mean episode rewards or success rates) at the end of the training process might not exactly equal the results reported in Tab. 4.1 & 4.2, which are obtained by batch evaluations using the model checkpoints.

4.7 Ablation Studies

In this section, we perform several ablation studies to justify our design. In ablation experiments, we use 256 different levels for all Procgens experiments, as opposed to 1024 different levels in the main result section (Sec. 4.6.2), since obtaining results on fewer levels is faster.

4.7.1 Influence of Granularity in Task Partitioning

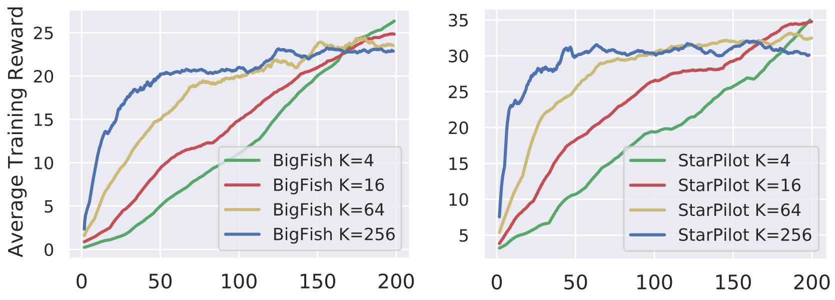


Figure 4.7. The average training returns on BigFish and StarPilot over 256 levels for $N_s = 256/K$ specialists trained by PPO. The generalist ($K = 256$) is more efficient early on but the specialists are more effective in the end.

A motivating observation for our framework is that training with more environment variations tends to be faster at the beginning but plateaus at sub-optimal performance. We show the evidence here, by varying the number of environment variations to train an RL agent. We pay attention to both the learning speed (efficiency) and the performance at the plateau (effectiveness).

On BigFish and StarPilot in Procgen, we use 256 levels generated with seeds from 2000 to 2256. Note that in Procgen, each seed guarantees a fixed and distinct level, i.e., environment variation. We choose $K = \{4, 16, 64, 256\}$ for the number of levels per specialist. Notice that $K = 256$ is equivalent to training a single generalist for all levels. For each K, we evenly and randomly distribute the 256 training levels to $N_s = 256/K$ specialists so that each specialist only learns on its distinctive set of levels. Each of the N_s specialists has a budget of 200 million/N_s samples to train the PPO so that the total sample budget is fixed for a fair comparison of sample complexity. We use the default network architecture and hyper-parameters for PPO in the Procgen paper, except that we reduce the number of parallel threads from 64 to 16 for better sample efficiency for specialists of $K = \{4, 16\}$. We plot the training curve as the average of the $N_s = 256/K$

specialists and scale it horizontally so that it reflects the total number of samples in a way same as the generalist's training curve.

The results in Fig. 4.7 clearly suggest our findings. In short, when trained from scratch, training a generalist enjoys a more efficient early learning process; in contrast, later on, training specialists with smaller variations is more effective in achieving better performance without plateauing early.

4.7.2 Influence of the Timing of Specialist Training

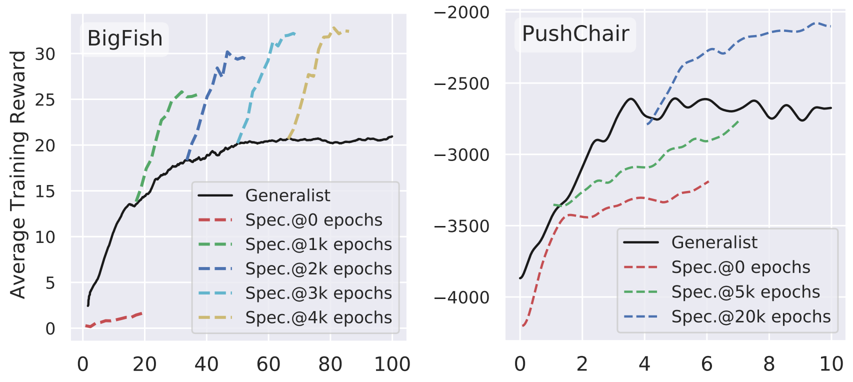


Figure 4.8. Training curves of specialists launched from the generalist trained with different numbers of epochs. Starting specialist training after the generalist's performance plateaus is more sample-efficient.

In Sec. 4.5.2, we mentioned that the timing to start specialist training plays a crucial role in the efficiency and effectiveness of our framework. In this section, we show further evidence. We launch specialist training at different stages of the generalist's training curve and present results in Fig. 4.8. We also conduct an experiment where specialists are trained from scratch instead of initiated from a generalist checkpoint (this corresponds to "Spec @ 0 epochs" in the figure). We use 64 specialist agents for BigFish training and 8 specialist agents for PushChair training.

We observe that when the generalist is still in the early stages of fast learning and has not reached a performance plateau, launching specialist training results in worse sample complexity. In particular, training specialists from scratch as in previous works

[163, 52] leads to inefficient and ineffective learning. On the other hand, after the generalist reaches a performance plateau, specialist training has very close efficiency and efficacy regardless of when the training is launched, as shown in the nearly parallel specialist curves in BigFish after the generalist has been trained for 2k epochs. Therefore, a good strategy for specialist training is to launch it as soon as the generalist reaches a performance plateau, which we adopt in Algorithm 1.

4.7.3 Tuning and Evaluation of Plateau Criteria $\mathcal{H}$

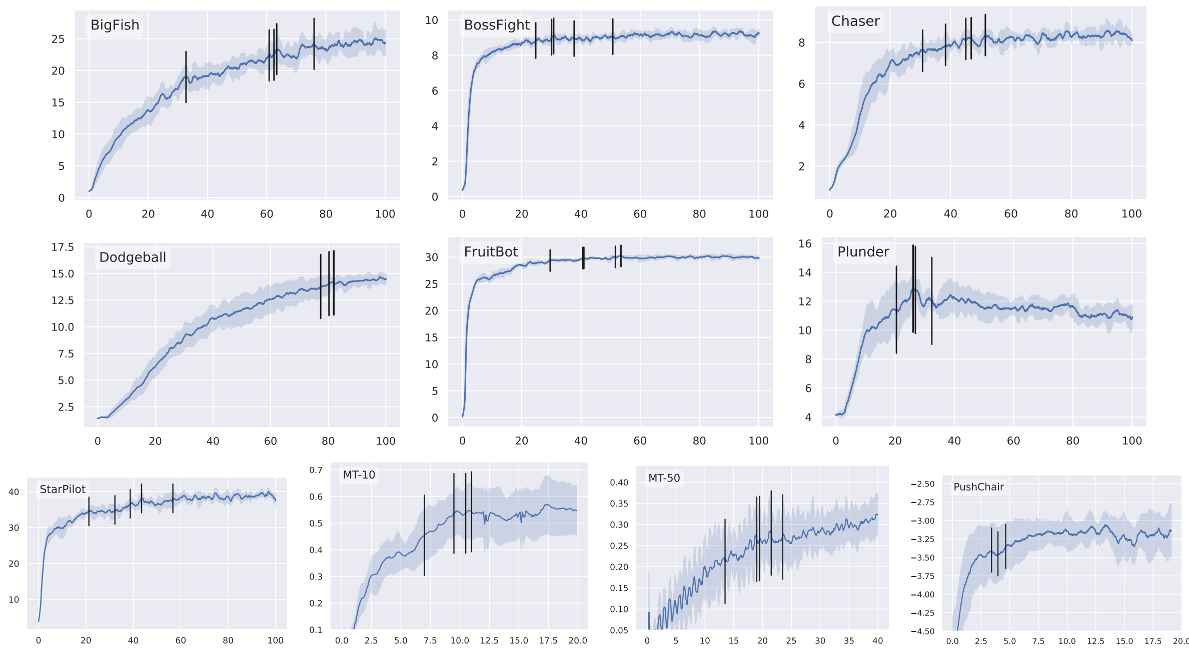


Figure 4.9. Qualitative evaluation of criteria $\mathcal{H}$. Vertical black lines indicate where $\mathcal{H}(t) = 1$ for the first time in each run of 5 runs for Procgen and Meta-World and 3 runs for PushChair in ManiSkill. The y-axis is raw episode rewards for Procgen and ManiSkill (whose unit is 1000) and the average success rate for Meta-World. The x-axis is a million steps.

Due to the different training dynamics of various RL algorithms on the benchmarks, the plateau criteria $\mathcal{H}$ does require some hyper-parameter tuning. However, we find it relatively robust and easy to tune. It turns out that starting specialist learning at a time 5% of the total training budget earlier or later than the one suggested by $\mathcal{H}$ achieves very

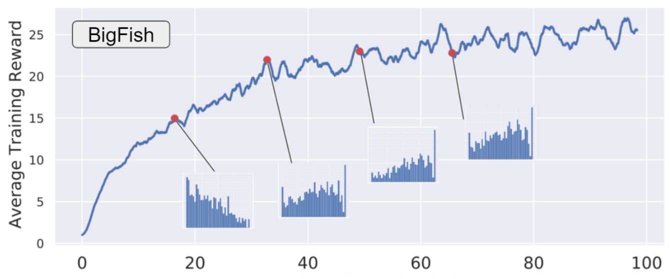


Figure 4.10. Training returns of PPO on BigFish with 256 levels and performance histograms over the withheld levels at four generalist checkpoints during training. Policy learning on BigFish achieves good progress in a large portion of levels but slow to no improvements on the remaining.

similar final results. Nevertheless, if $\mathcal{H}$ is tuned too aggressively (too early), we lose some efficiency from the initial generalist learning; if it is tuned too conservatively (too late), the generalist plateaus too long (i.e., less efficiency) and could become too certain about its decisions, rendering it harder to be fine-tuned with specialists' knowledge. In addition, in practice, we do not consider the first 15% to avoid the cases where the generalist gets stuck at the very beginning (i.e., a "slow start" generalist). We also do not consider the last 15% to leave a sufficient amount of samples for launching specialist training and using specialists' demonstrations to guide generalist training.

Here we verify the effectiveness of our criteria $\mathcal{H}$ by providing a qualitative evaluation on Procgen, ManiSkill, and Meta-World. Specifically, we use a smoothing kernel size of 200 epochs for Procgen & ManiSkill and 10 for Meta-World; the window size is $W = 600$ for Procgen, $W = 2000$ for ManiSkill and $W = 10$ for Meta-World; $\epsilon = 0.03$ for Procgen, $\epsilon = 0.01$ for Meta-World and $\epsilon = 0.05k$ for ManiSkill. For each of the 5 runs in Procgen environments and one run in PushChair from ManiSkill, we mark the first timestamp t where our proposed criteria $\mathcal{H}(t)$ is evaluated as 1 with the black vertical lines. Areas close to the vertical lines are generally suitable for starting specialist training. The results are visualized in Fig. 4.9.

4.7.4 Diagnosis into Generalist Performance at Plateau

In Sec. 4.5.2, we discussed the strategy to assign environment variations to specialists. Specifically, we launch specialist training only on the N_s variants with the lowest training performance evaluated using the generalist. Our strategy is based on the observation that the generalist can solve some environment variations reasonably well, but performs poorly on others. We visualize the performance of the generalist on BigFish from Procgen in Fig. 4.10, where the PPO agent makes quick progress on most of the levels, with poor performance only on a portion of variations at the time of plateau. We find this quite common for environments in Procgen, except for Plunder where PPO struggles in a majority of levels.

4.7.5 Influence of Specialist-to-Generalist Merging Algorithms

In Sec. 4.5.3, we mentioned the importance of selecting the algorithm for consolidating specialists' experience for generalist learning. We conduct experiments on BigFish of Procgen, where we set the number of specialists to be 75. We compare GSL+BC, GSL+DAPG, and DnC [52], a classic divide-and-conquer RL work that uses BC. As a baseline, we also show the performance of jointly training a PPO on all variations. However, for DnC RL, we find that, unlike any other setups, it has quadratic computational complexity w.r.t. the number of specialists, rendering it computationally intractable, so

Table 4.3. Comparison to baselines and other design choices. DnC RL requires synchronized training across all specialists with a quadratic computational complexity w.r.t. the number of specialists, and we find it hard to scale up to our setup of 75 specialists as in GSL. Using GSL with BC harms the performance. Our design works better than all other choices.

	BigFish (train)	BigFish (test)
PPO (generalist only)	24.6±0.7	24.3±1.1
DnC RL	-	-
GSL with BC	25.3±0.4	22.1±0.9
GSL with DAPG	**31.1±0.8**	**30.0±0.5**

we skip this method. As shown in Table. 4.3, using BC in our GSL framework does not help to outperform a single PPO generalist. One possible reason is that GSL does not restrict specialists to be close to the generalist as in DnC, so specialists' demonstrations can be inconsistent, posing learning difficulties for BC. On the other hand, our GSL with DAPG can outperform a single PPO by a large margin. Additionally, we sample 1,000 novel levels from BigFish to test the generalization performance of our models. Results show that GSL+DAPG stays the best for generalization.

4.8 Implementation Details

4.8.1 Illustrative example

In our illustrative example, we train the generalist and the specialists using PPO and use DAPG + PPO to resume generalist training with demos from specialists. We summarize the hyperparameters in Tab. 4.4 & 4.5.

Table 4.4. Hyperparameters for DAPG and PPO in our illustrative example

Hyperparameters	Value
Optimizer	Adam
Learning rate	3×10^{-4}
Discount (γ)	0.95
λ in GAE	0.97
PPO clip range	0.2
Coefficient of the entropy loss term of PPO c_{ent}	0.01
Coefficient of the entropy loss term of PPO c_{ent} (during DAPG)	0.01
Number of hidden layers (all networks)	2
Number of hidden units per layer	256
Number of threads for collecting samples	5
Number of samples per PPO step	10^4
Number of samples per minibatch	2000
Nonlinearity	ReLU
Total Simulation Steps	5×10^6
Number of environment variations	5
Environment horizon	150

Table 4.5. Hyperparameters of GSL in our illustrative example.

Hyperparameters	Value
Number of specialists N_s	5
Number of lowest performing environment variations assigned to specialists N_{lenv}	5
Number of environment variations per specialist K	1
Number of demos generated from specialists (epochs $\times$ time steps per epoch $\times$ N_s) N_D^s	$10 \times 150 \times 5$
Number of demo samples from generalist N_D^g	N/A
Sliding window size (epochs) for the plateau criteria W	50
Number of samples for training each specialist N_{sample}	$500K$
Number of samples for DAPG	$1M$
Margin in the plateau criteria ϵ	0.01

4.8.2 Procgen

In Procgen, we train the generalist and the specialists using PPO/PPG (which is an extension of PPO), and use DAPG + PPO/PPG to resume generalist training with demos from specialists. We summarize the hyperparameters in Tab. 4.6, 4.7 and 4.8.

Table 4.6. The hyperparameters of PPO and DAPG for Procgen experiments.

Hyperparameters	Value
Optimizer	Adam
Learning rate	5×10^{-4}
Discount (γ)	0.999
λ in GAE	0.95
PPO clip range	0.2
Coefficient of the entropy loss term of PPO c_{ent}	0.01
Coefficient of the entropy loss term of PPO c_{ent} (during DAPG)	0.05
Number of threads for collecting samples	64
Number of samples per PPO epoch	256×64
Number of samples per minibatch	1024
Nonlinearity	ReLU
Total Simulation Steps	10^8

Table 4.7. Additional hyperparameters of PPG for Procgen experiments.

Hyperparameters	Value
Number of policy update epochs in each policy phase `n_pi`	32
Number of auxiliary epochs in each auxiliary phase `n_aux_epochs`	6
Coefficient of the entropy loss term of PPO c_{ent}	0.0
Coefficient of the entropy loss term of PPO c_{ent} (during DAPG)	0.01

Table 4.8. The hyperparameters of GSL for experiments on Procgen.

Hyperparameters	Value
Number of specialists N_s	75
Number of lowest performing environment variations assigned to specialists N_{lenv}	300
Number of environment variations per specialist K	4
Number of demos generated (epochs × time steps per epoch $\times N_s$) N_D^s	$256 \times 32 \times 75$
Number of demos generated (epochs × time steps per epoch × number of env. var.) N_D^g	$256 \times 8 \times 724$
Sliding window size (epochs) for the plateau criteria W	600
The threshold for filtering demos (in terms of the normalized score) τ	0.15
Number of samples for training each specialist N_{sample} (PPO)	$16 \times 256 \times 64$
Number of samples for training each specialist N_{sample} (PPG)	$20 \times 256 \times 64$
Number of samples for DAPG (for PPO)	$50 \times 256 \times 64$
Number of samples for DAPG (for PPG)	$800 \times 256 \times 64$
Margin in the plateau criteria ϵ	0.03

Other implementation details

For all environments, we use seeds (levels) from 1000 to 2023 for training and from 100000 to 100999 for testing. When training the specialists, we change the number of parallel environments in PPO from 64 to 16 for a slightly better sample efficiency, and we change both `n_pi` and `n_aux_epochs` to 4 from PPG for a similar reason. We find that increasing the coefficient c_{ent} for the entropy regularization loss during DAPG can help improve both the optimization and generalization performance for Procgen. For the Plunder environment (with PPO as the baseline), we have observed poor generalist training performance across a majority of levels. We therefore set $N_s = 1024$, i.e., we train specialists on all levels. We also change the number of samples used in DAPG to $200 \times 256 \times 64$ since there are more demos collected from the specialists than in other tasks. Moreover, in Plunder we set c_{ent} to 0.1 during DAPG.

4.8.3 Meta-World

For both MT-10 and MT-50 from Meta-World, we train the generalist and the specialists using PPO (with a policy network of two hidden layers, each of hidden size 32), and use DAPG + PPO to resume generalist training with demos from specialists. We summarize the hyperparameters in Tab. 4.10 & 4.9.

Table 4.9. The hyperparameters of GSL for experiments on Meta-World.

Hyperparameters	Value
Number of lowest performing environment variations assigned to specialists N_{lenv}	Varied
Number of specialists N_s	N_{lenv}
Number of environment variations per specialist K	1
Number of demos generated (time steps per env. variations $\times N_s$) N_D^s	$10^5 \times N_s$
Number of demos generated (time steps per env. var. $\times$ # of remaining env. var. N_r) N_D^g	$10^5 \times N_r$
Sliding window size (epochs) for the plateau criteria W	10
The threshold for filtering demos (in terms of success rate) τ	1.0 (successful)
Number of samples for training each specialist N_{sample}	3000×700
Number of samples for DAPG (MT-10)	2×10^6
Number of samples for DAPG (MT-50)	6×10^6
Margin in the plateau criteria ϵ	0.01

Table 4.10. The hyperparameters of PPO and DAPG for Meta-World experiments.

Hyperparameters	Value
Optimizer	Adam
Learning rate	2.5×10^{-4}
Discount (γ)	0.99
λ in GAE	0.95
Minimum std. of the Gaussian policy `min_std`	0.5
Maximum std. of the Gaussian policy `max_std`	1.5
PPO clip range	0.2
Coefficient of the entropy loss term of PPO c_{ent} (MT-10)	0.005
Coefficient of the entropy loss term of PPO c_{ent} (MT-50)	0.05
Number of threads for collecting samples (MT-10)	10
Number of threads for collecting samples (MT-50)	50
Number of samples per PPO epoch	10^5
Number of samples per minibatch	32
Nonlinearity	ReLU
Total Simulation Steps (MT-10)	2×10^7
Total Simulation Steps (MT-50)	4×10^7

Other implementation details

We find that in MT-10/50, there always exist some tasks (i.e., environment variations) that the generalist agent performs extremely poorly after the initial learning phase. We, therefore, use a threshold of the success rate of 0.5 for MT-10 and 0.2 for MT-50 to select the N_{lenv} (which varies across different runs) lowest performing env. variations and correspondingly launch N_{lenv} specialists. During specialist training, we reduce the number of samples per PPO epoch from 10^5 to 3×10^3 to improve the sample efficiency (since each specialist now only learns to solve one environment variation).

Table 4.11. The hyperparameters of SAC and GAIL+SAC for ManiSKill

Hyperparameters	Value
Optimizer	Adam
Learning rate	3×10^{-4}
Discount (γ)	0.95
Replay buffer size (γ)	2×10^{6}
Number of threads for collecting samples	4
Number of samples per minibatch	200
Nonlinearity	ReLU
Target smoothing coefficient(τ)	0.005
Target update interval	1
Q, π update frequency	4 updates per 64 online samples
GAIL discriminator update frequency	5 updates per 100 policy updates
Total Simulation Steps	2×10^{7}

4.8.4 ManiSkill

For ManiSkill experiments, we adopt the same PointNet + Transformer over object segmentation architecture as in the original paper. We proportionally downsample point cloud observations to 1200 points following the same strategy in the original paper. We train the generalist and the specialists using SAC and we resume generalist training using GAIL + SAC with demos from specialists. Notice that each specialist only focuses on one chair model in the PushChair task. We use the hyperparameters listed in Tab. 4.11, inspired by the implementations of [147]. We also show hyperparameters for GSL in Tab. 4.12.

4.9 Conclusion

Generalization in RL usually requires training in diverse environments. In this work, we develop a simple yet effective framework to solve RL problems that involve a large number of environment variations. The framework is a meta-algorithm that turns a pair of RL algorithms and learning-from-demonstrations algorithms into a more powerful RL algorithm. By analysis of prototypical cases, we identify that the catastrophic

Table 4.12. The hyperparameters of GSL for experiments on ManiSkill.

Hyperparameters	Value
Number of specialists N_s	8
Number of lowest performing env. variations assigned to specialists N_{lenv}	8
Number of environment variations per specialist K	1
Number of demos generated from specialists N_D^s	$200 \times 300 \times 8$
Number of demos generated from generalist N_D^g	N/A
Sliding window size (epochs) for the plateau criteria W	2000
Threshold for filter the demos τ	-3.5×10^3
Number of samples for training each specialist N_{sample}	1.5×10^6
Number of samples used in GAIL+SAC	2×10^6
Margin used in the plateau criteria ϵ	50

ignorance and forgetting of neural networks pose significant challenges to RL training in environments with many variations, and may cause the agent to reach performance plateau at a sub-optimal level. We show that introducing specialists to train in subsets of environments can effectively escape from this performance plateau and reach a high reward. Design choices that are crucial yet unknown in the literature must be taken care of for the success of our framework. Empirically, our framework achieves high efficiency and effectiveness by improving modern RL algorithms on several popular and challenging benchmarks.

Acknowledgement

Chapter 4, in full, is a reprint of the material published in the 2022 International Conference on Machine Learning (ICML): “Improving Policy Optimization with Generalist-Specialist Learning” (Zhiwei Jia, Xuanlin Li, Zhan Ling, Shuang Liu, Yiran Wu, Hao Su). The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] 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 preprint arXiv:2204.01691*, 2022.

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

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

[4] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

[5] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[6] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

[7] Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR, 2022.

[8] Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR, 2022.

[9] Timo Bram, Gino Brunner, Oliver Richter, and Roger Wattenhofer. Attentive multi-task deep reinforcement learning. *arXiv preprint arXiv:1907.02874*, 2019.

[10] Kiante Brantley, Wen Sun, and Mikael Henaff. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2019.

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

[12] 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. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[13] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[14] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017.

[15] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[16] Jonathan Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data with partial coverage. *Advances in Neural Information Processing Systems*, 34:965–979, 2021.

[17] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *The International Conference on Learning Representations (ICLR)*, 2020.

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

[19] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable spatial planning using transformers. In *International Conference on Machine Learning*, pages 1484–1495. PMLR, 2021.

[20] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12875–12884, 2020.

[21] Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12538–12547, 2019.

[22] 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. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[23] Linghao Chen, Yuzhe Qin, Xiaowei Zhou, and Hao Su. Easyhec: Accurate and automatic hand-eye calibration via differentiable rendering and space exploration. *arXiv preprint arXiv:2305.01191*, 2023.

[24] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *International Conference on Learning Representations*, 2018.

[25] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object reorientation. In *Conference on Robot Learning*, pages 297–307. PMLR, 2022.

[26] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15750–15758, 2021.

[27] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[28] Ron Chrisley. Embodied artificial intelligence. *Artificial intelligence*, 149(1):131–150, 2003.

[29] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[30] Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*, 2020.

[31] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[32] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[33] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[34] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR, 2021.

[35] Carlos G Correa, Mark K Ho, Frederick Callaway, Nathaniel D Daw, and Thomas L Griffiths. Humans decompose tasks by trading off utility and computational cost. *PLOS Computational Biology*, 19(6):e1011087, 2023.

[36] Wojciech Czarnecki, Siddhant Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Nicolas Heess, Simon Osindero, and Razvan Pascanu. Mix & match agent curricula for reinforcement learning. In *International Conference on Machine Learning*, pages 1087–1095. PMLR, 2018.

[37] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–10, 2018.

[38] Sudeep Dasari and Abhinav Gupta. Transformers for one-shot visual imitation. In *Conference on Robot Learning*, pages 2071–2084. PMLR, 2021.

[39] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[40] Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient rl with rich observations via latent state decoding. In *International Conference on Machine Learning*, pages 1665–1674. PMLR, 2019.

[41] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[42] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*, 2022.

[43] Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2020.

[44] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016.

[45] Adam Fishman, Adithyavairan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. *arXiv preprint arXiv:2210.12209*, 2022.

[46] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *Conference on Robot Learning*, pages 967–977. PMLR, 2023.

[47] 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 *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[48] Peter Florence, Lucas Manuelli, and Russ Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

[49] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[50] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

[51] David V Gealy, Stephen McKinley, Brent Yi, Philipp Wu, Phillip R Downey, Greg Balke, Allan Zhao, Menglong Guo, Rachel Thomasson, Anthony Sinclair, et al. Quasi-direct drive for low-cost compliant robotic manipulation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 437–443. IEEE, 2019.

[52] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

[53] Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[54] Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill mobile manipulation for object rearrangement. *arXiv preprint arXiv:2209.02778*, 2022.

[55] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiaing Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023.

[56] Abhinav Gupta, Adithyavairavan Murali, Dhiraj Gandhi, and Lerrel Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. *arXiv preprint arXiv:1807.07049*, 2018.

[57] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[58] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[59] Brent Harrison, Upol Ehsan, and Mark O Riedl. Guiding reinforcement learning exploration using natural language. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1956–1958, 2018.

[60] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[61] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[62] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[63] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

[64] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[65] Ronald A Howard. Dynamic programming and markov processes. 1960.

[66] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[67] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

[68] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *arXiv preprint arXiv:1910.12911*, 2019.

[69] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28:2017–2025, 2015.

[70] Stephen James and Andrew J Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *IEEE Robotics and Automation Letters*, 7(2):1612–1619, 2022.

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

[72] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[73] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[74] Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1238–1247, 2018.

[75] Zhiwei Jia, Xuanlin Li, Zhan Ling, Shuang Liu, Yiran Wu, and Hao Su. Improving policy optimization with generalist-specialist learning. In *International Conference on Machine Learning*, pages 10104–10119. PMLR, 2022.

[76] Zhiwei Jia, Kaixiang Lin, Yizhou Zhao, Qiaozi Gao, Govind Thattai, and Gaurav Sukhatme. Learning to act with affordance-aware multimodal neural slam. *arXiv preprint arXiv:2201.09862*, 2022.

[77] Zhiwei Jia, Pradyumna Narayana, Arjun R Akula, Garima Pruthi, Hao Su, Sugato Basu, and Varun Jampani. Kafa: Rethinking image ad understanding with knowledge-augmented feature adaptation of vision-language models. *arXiv preprint arXiv:2305.18373*, 2023.

[78] Zhiwei Jia and Hao Su. Information-theoretic local minima characterization and regularization. In *International Conference on Machine Learning*, pages 4773–4783. PMLR, 2020.

[79] Zhiwei Jia, Bodi Yuan, Kangkang Wang, Hong Wu, David Clifford, Zhiqiang Yuan, and Hao Su. Semantically robust unpaired image translation for data with unmatched semantics statistics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14273–14283, 2021.

[80] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

[81] 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. *arXiv preprint arXiv:1806.10293*, 2018.

[82] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International conference on machine learning*, pages 2469–2478. PMLR, 2018.

[83] R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, oct 2012.

[84] Byeonghwi Kim, Suvaansh Bhambri, Kunal Pratap Singh, Roozbeh Mottaghi, and Jonghyun Choi. Agent with the big picture: Perceiving surroundings for interactive instruction following. In *Embodied AI Workshop CVPR*, 2021.

[85] David Inkyu Kim and Gaurav S Sukhatme. Interactive affordance map building for a robotic task. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4581–4586. IEEE, 2015.

[86] Mate Kisantal, Zbigniew Wojna, Jakub Murawski, Jacek Naruniec, and Kyunghyun Cho. Augmentation for small object detection. *arXiv preprint arXiv:1902.07296*, 2019.

[87] 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 preprint arXiv:1712.05474*, 2017.

[88] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pages 5774–5783. PMLR, 2021.

[89] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

[90] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[91] Vikash Kumar and Emanuel Todorov. Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 657–663. IEEE, 2015.

[92] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M Pawan Kumar. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122*, 2022.

[93] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pages 143–156. PMLR, 2017.

[94] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[95] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. Igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021.

[96] Zhengzhong Liang, Steven Bethard, and Mihai Surdeanu. Explainable multi-hop verbal reasoning through internal monologue. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.

[97] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[98] Fangchen Liu, Hao Liu, Aditya Grover, and Pieter Abbeel. Masked autoencoding for scalable and generalizable decision making. *arXiv preprint arXiv:2211.12740*, 2022.

[99] William S Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[100] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *arXiv preprint arXiv:1908.02265*, 2019.

[101] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

[102] Zhao Mandi, Fangchen Liu, Kimin Lee, and Pieter Abbeel. Towards more generalizable one-shot visual imitation learning. *arXiv preprint arXiv:2110.13423*, 2021.

[103] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[104] So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*, 2021.

[105] Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. *arXiv preprint arXiv:1809.00786*, 2018.

[106] Tongzhou Mu, Jiayuan Gu, Zhiwei Jia, Hao Tang, and Hao Su. Refactoring policy for compositional generalizability using self-supervised object proposals. *arXiv preprint arXiv:2011.00971*, 2020.

[107] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. *arXiv preprint arXiv:2107.14483*, 2021.

[108] Tushar Nagarajan and Kristen Grauman. Learning affordance landscapes for interaction exploration in 3d environments. *Advances in Neural Information Processing Systems*, 33:2005–2015, 2020.

[109] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[110] Van-Quang Nguyen, Masanori Suganuma, and Takayuki Okatani. Look wide and interpret twice: Improving performance on interactive instruction-following tasks. In *30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 923–930. International Joint Conferences on Artificial Intelligence, 2021.

[111] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[112] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[113] 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. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[114] Yingwei Pan, Yehao Li, Yiheng Zhang, Qi Cai, Fuchen Long, Zhaofan Qiu, Ting Yao, and Tao Mei. Silver-bullet-3d at maniskill 2021: Learning-from-demonstrations and heuristic rule-based methods for object manipulation. *arXiv preprint arXiv:2206.06289*, 2022.

[115] Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic transformer for vision-and-language navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15942–15952, 2021.

[116] Keiran Paster, Sheila McIlraith, and Jimmy Ba. You can't count on luck: Why decision transformers fail in stochastic environments. *arXiv preprint arXiv:2205.15967*, 2022.

[117] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[118] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4561–4570, 2019.

[119] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021.

[120] Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In *Conference on Robot Learning*, pages 2279–2291. PMLR, 2021.

[121] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[122] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.

[123] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[124] William Qi, Ravi Teja Mullapudi, Saurabh Gupta, and Deva Ramanan. Learning to move with affordance maps. In *International Conference on Learning Representations*, 2019.

[125] Yuzhe Qin, Hao Su, and Xiaolong Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *arXiv preprint arXiv:2204.12490*, 2022.

[126] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[127] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE, 2018.

[128] Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. *arXiv preprint arXiv:2102.10330*, 2021.

[129] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

[130] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*, 2019.

[131] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[132] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[133] Santhosh K Ramakrishnan, Dinesh Jayaraman, and Kristen Grauman. An exploration of embodied visual exploration. *International Journal of Computer Vision*, 129(5):1616–1649, 2021.

[134] Tabish Rashid, Bei Peng, Wendelin Boehmer, and Shimon Whiteson. Optimistic exploration even with a pessimistic initialisation. *arXiv preprint arXiv:2002.12174*, 2020.

[135] Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[136] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[137] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[138] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[139] Fumihiro Sasaki and Ryota Yamashina. Behavioral cloning from noisy demonstrations. In *International Conference on Learning Representations*, 2020.

[140] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9339–9347, 2019.

[141] Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv preprint arXiv:1904.11455*, 2019.

[142] Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991.

[143] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[144] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[145] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[146] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *arXiv preprint arXiv:2206.11251*, 2022.

[147] Hao Shen, Weikang Wan, and He Wang. Learning category-level generalizable object manipulation policy via generative adversarial self-imitation learning from demonstrations. *arXiv preprint arXiv:2203.02107*, 2022.

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

[149] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[150] 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.

[151] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[152] Kunal Pratap Singh, Suvaansh Bhambri, Byeonghwi Kim, Roozbeh Mottaghi, and Jonghyun Choi. Factorizing perception and policy for interactive instruction following. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1888–1897, 2021.

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

[154] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. *arXiv preprint arXiv:2102.06177*, 2021.

[155] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9767–9779. PMLR, 18–24 Jul 2021.

[156] Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav Sukhatme. Embodied bert: A transformer model for embodied, language-guided visual task completion. *Learning-to-Learn through Interaction (NILLI) workshop, EMNLP*, 2021.

[157] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International conference on machine learning*, pages 3309–3318. PMLR, 2017.

[158] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[159] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34, 2021.

[160] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *31st Conference on Neural Information Processing Systems (NIPS)*, volume 30, pages 1–18, 2017.

[161] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[162] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

[163] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

[164] Guy Tennenholtz, Assaf Hallak, Gal Dalal, Shie Mannor, Gal Chechik, and Uri Shalit. On covariate shift of latent confounders in imitation and reinforcement learning. *arXiv preprint arXiv:2110.06539*, 2021.

[165] Jesse Thomason, Michael Murray, Maya Cakmak, and Luke Zettlemoyer. Vision-and-dialog navigation. In *Conference on Robot Learning*, pages 394–406. PMLR, 2020.

[166] Yusuke Urakami, Alec Hodgkinson, Casey Carlin, Randall Leu, Luca Rigazio, and Pieter Abbeel. Doorgym: A scalable door opening environment and baseline agent. *arXiv preprint arXiv:1908.01887*, 2019.

[167] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[168] Quan Vuong, Yuzhe Qin, Runlin Guo, Xiaolong Wang, Hao Su, and Henrik Christensen. Single rgb-d camera teleoperation for general robotic manipulation. *arXiv preprint arXiv:2106.14396*, 2021.

[169] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[170] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[171] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6659–6668, 2019.

[172] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019.

[173] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827. PMLR, 2019.

[174] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020.

[175] Danfei Xu, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. *arXiv preprint arXiv:2011.08424*, 2020.

[176] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3795–3802. IEEE, 2018.

[177] Jing Xu, Zhimin Hou, Zhi Liu, and Hong Qiao. Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies. *arXiv preprint arXiv:1904.05240*, 2019.

[178] Mengjiao Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning. *arXiv preprint arXiv:2205.10816*, 2022.

[179] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

[180] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

[181] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[182] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[183] Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, pages 537–546. PMLR, 2022.

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

[185] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[186] Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block mdps. In *International Conference on Machine Learning*, pages 11214–11224. PMLR, 2020.

[187] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[188] Xiaoshuai Zhang, Rui Chen, Fanbo Xiang, Yuzhe Qin, Jiayuan Gu, Zhan Ling, Minghua Liu, Peiyu Zeng, Songfang Han, Zhiao Huang, et al. Close the visual domain gap by physics-grounded active stereovision depth sensor simulation. *arXiv preprint arXiv:2201.11924*, 2022.

[189] Yichi Zhang and Joyce Chai. Hierarchical task learning from language instructions with unified transformers and self-monitoring. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4202–4213, Online, August 2021. Association for Computational Linguistics.

[190] Yizhou Zhao, Kaixiang Lin, Zhiwei Jia, Qiaozi Gao, Govind Thattai, Jesse Thomason, and Gaurav S Sukhatme. Luminous: Indoor scene generation for embodied ai challenges. *arXiv preprint arXiv:2111.05527*, 2021.

[191] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, 2022.