UNIVERSITY OF CALIFORNIA SAN DIEGO

Modeling the Physical World for Data-Centric Robotics

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

in

Computer Science

by

Tongzhou Mu

Committee in charge:

Professor Hao Su, Chair
Professor Sanjoy Dasgupta
Professor Sicun Gao
Professor Michael Yip

2024

The Dissertation of Tongzhou Mu is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

DEDICATION

I dedicate this dissertation to my grandmother, Ruixue Ye, and my parents, Jianfeng Zhou and Kai Mu, for their unconditional love and endless support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

After more than seven years at UC San Diego, including my master's and Ph.D., I have finally completed this wonderful journey. For me, it has been both a grind and an adventure. I remember when I decided to pursue a Ph.D., I told myself that in the worst-case scenario, I would regret it for five years, but if I didn't pursue it, I might regret it for the rest of my life. So, I went ahead and did it, and now I can say I have no regrets. This is not because of the papers I published or the achievements I attained during my Ph.D., but rather because of the failures, the grind, and the struggles along the way. They have made me a stronger person from within.

This dissertation would not have been possible without the support and contributions of numerous individuals. I want to express my sincere gratitude to my advisors, collaborators, friends, and everyone who helped me get through these years.

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Hao Su, for his invaluable guidance, unwavering support, and constant encouragement throughout my Ph.D. journey. His insightful feedback, stimulating discussions, and steadfast belief in my abilities have shaped my research and helped me grow as a researcher. I began working under Professor Hao Su's supervision when I was a master's student. He guided me onto the path of research, provided me with the opportunity to pursue a Ph.D., and influenced my research taste. There is a Chinese idiom that says, "there are many swift horses, but very few who can spot them." I would not be here without Professor Hao Su's trust.

I am also deeply grateful to my committee members, Professor Sanjoy Dasgupta, Professor Sicun Gao, and Professor Michael Yip, for their valuable time and insightful comments. I had the privilege of taking classes taught by each of these professors in the early years of my Ph.D., and I feel fortunate to have explored topics outside my own research under their guidance.

I would also like to thank Dr. Jiaolong Yang for mentoring me at Microsoft Research Asia during my undergraduate years. Before my internship at MSRA, I knew nothing about how to conduct research, and it was Dr. Jiaolong Yang who opened the door to research for me. Additionally, it was Dr. Jiaolong Yang who mentioned Professor Hao Su's name to me. Without

Jiaolong, I would not have had such a wonderful journey from the very beginning.

I have been fortunate to make friends with and receive support from many brilliant collaborators over the years, and I would like to extend my heartfelt thanks to all of my collaborators. In particular, I want to thank Jiayuan Gu, Zhiwei Jia, Zhan Ling, Fangchen Liu, Minghua Liu, Xingchao Liu, and Stone Tao for being my friends and collaborators throughout different phases of my Ph.D. career. Their expertise and collaborative spirit have been invaluable. I also want to thank all members of the SU Lab for creating a stimulating and supportive research environment. I have greatly benefited from the discussions, collaborations, and friendships within the lab.

I am also very grateful to my friends Lin Chu, Kesong Feng, Lifeng Huang, Zhihao Wen, and Tong Yi for their unconditional emotional support throughout these years, especially during the COVID-19 period. Their remote companionship helped me get through the most difficult times during my Ph.D. journey.

During my years at UC San Diego, I attended numerous events at UCSD Recreation and UCSD Outback Adventure, which greatly contributed to my physical and emotional health. I want to express my gratitude to all the class instructors and Outback guides, especially Jessica, Lauren, Kojack, Jessie, Sean, Jillian, Julianna, Ray, and Savanna. They have been a bright spot in my otherwise challenging Ph.D. life.

Finally, I express my heartfelt gratitude to my family for their unwavering love, support, and encouragement. Their belief in me has been a constant source of motivation throughout this journey. To my parents, Jianfeng Zhou and Kai Mu, thank you for your endless love and support. This dissertation is dedicated to you.

Chapter 1, in full, is taken from "ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations" in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track 2021 by Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, Hao Su. The dissertation author was a primary investigator and author of this paper.

Chapter 2, in full, is taken from "DrS: Learning Reusable Dense Rewards for Multi-Stage

Tasks" in International Conference on Learning Representations (ICLR) 2024 by Tongzhou Mu, Minghua Liu, Hao Su. The dissertation author was a primary investigator and author of this paper.

Chapter 3, in full, is taken from "Abstract-to-Executable Trajectory Translation for One-Shot Task Generalization" in International Conference on Machine Learning (ICML) 2023 by Stone Tao, Xiaochen Li, Tongzhou Mu, Zhiao Huang, Yuzhe Qin, Hao Su. The dissertation author was an author of this paper.

Chapter 4, in full, is taken from "Policy Decorator: Model-Agnostic Online Refinement for Large Policy Model" by Xiu Yuan, Tongzhou Mu, Stone Tao, Yunaho Fang, Michael Zhang, Hao Su. It has been submitted for publication of the material as it may appear in the International Conference on Learning Representations (ICLR) 2025. The dissertation author was a primary investigator and author of this paper.

VITA

2017 Bachelor of Engineering in Computer Science, Zhejiang University, China.

2019 Master of Science in Computer Science, University of California San Diego, USA.

2024 Doctor of Philosophy in Computer Science, University of California San Diego, USA.

ABSTRACT OF THE DISSERTATION

Modeling the Physical World for Data-Centric Robotics

by

Tongzhou Mu

Doctor of Philosophy in Computer Science

University of California San Diego, 2024

Professor Hao Su, Chair

The development of a general-purpose robot capable of performing diverse tasks in home and office environments remains a central challenge in robotics. While breakthroughs in foundation models for language and vision have demonstrated the potential of scaling data and computation, robotics faces unique hurdles due to its reliance on physical-world data, such as 3D geometry, dynamics, and spatiotemporal information. These challenges create a significant data bottleneck, limiting progress toward general-purpose robots. This dissertation advocates for a data-centric approach to robotics, focusing on three core components: 1) building environments for scalable data collection, 2) developing efficient methods for collecting diverse robotic demonstrations, and 3) leveraging this data to learn robust and generalizable robotic

policies.

To address these challenges, this dissertation introduces ManiSkill, a large-scale simulation benchmark designed for learning and evaluating diverse robotic manipulation skills. ManiSkill features extensive 3D assets, diverse tasks, and large-scale demonstration datasets, providing a robust environment for data-centric research. For scalable demonstration collection, this work proposes DrS, a method for learning reusable dense rewards from sparse rewards and demonstrations, enabling efficient reinforcement learning for short-horizon tasks. Additionally, it presents TR2, a framework that generates demonstrations for long-horizon tasks by translating abstract trajectories into executable robot actions. Finally, this dissertation introduces Policy Decorator, a model-agnostic method for refining imitation learning-based policies through controlled online interactions, significantly improving performance while preserving smooth motion.

By integrating these contributions, this dissertation establishes a comprehensive framework for advancing data-centric robotics, addressing critical bottlenecks in data collection and policy learning, and paving the way toward general-purpose robotic systems.

Introduction

The vision of a general-purpose robot capable of reliably performing a wide range of tasks in home or office environments, akin to human capabilities, has long been regarded as the holy grail of robotics. However, the reality of robotics research presents significant challenges. Inspiration can be drawn from advancements in foundation models. Today, powerful language models such as GPT [143], Gemini [196], and Claude serve as chatbots, coding assistants, and more. Similarly, in computer vision, foundation models like SAM [91] for segmentation, Midjourney for text-to-image generation, and Sora for text-to-video generation have demonstrated remarkable capabilities. These models are notably more powerful and generalizable than prior neural networks. Their success largely stems from training on extensive large-scale internet datasets, underscoring the *importance of dataset scale as a critical factor in addressing complex tasks*.

Therefore, it is natural to consider applying this approach to robotics—scaling up datasets and computational resources with the hope of developing a general-purpose robot. However, this transition is far from straightforward. In 1988, Moravec wrote: "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility" [130]. This observation, known as Moravec's Paradox, highlights the challenges in directly transferring methodologies from perception and language research to robotics. A closer examination reveals fundamental differences: while NLP and CV operate as "Digital AI" within the digital domain, leveraging abundant text and image data, robotics represents "Physical AI," requiring interaction with and understanding of the physical world's complexities. This

necessitates data including 3D geometry, dynamics, and spatiotemporal information, posing a significant data bottleneck that hinders progress toward general-purpose robots.

These challenges are evident in recent robotics research. Efforts to develop robotic foundation models through imitation learning, leveraging large demonstration datasets, have gained significant attention [16, 19, 18, 5]. However, these models lag far behind their vision and language counterparts in terms of generalization and robustness. Their performance is critically dependent on the quantity, quality, and diversity of the demonstration data. Furthermore, initiatives to collect large-scale robot demonstration datasets, such as RT-1 [19] and Open X-Embodiment [33], demand substantial time and resources, requiring years of data collection through extensive human teleoperation, making the process costly. These considerations emphasize that data is central to the development of general-purpose robots. *A data-centric approach is essential to address this challenge effectively.*

This dissertation advocates for a data-centric approach to robotics, mirroring the successful paradigm in other AI domains. Data-centric robotics focuses on three core components: 1) building environments for robotics data collection; 2) developing efficient methods for collecting diverse robotic data, particularly demonstrations; and 3) utilizing this data to learn robust and generalizable robotic policies.

To generate sources of robotic data, Chapter 1 introduces ManiSkill, an interactive simulation environment designed to facilitate the learning and evaluation of diverse robotic skills. Significant advancements have been made in scaling up real-world robotic data collection [87, 29, 233, 163], while numerous simulation environments have also been developed for this purpose [110, 156, 127]. Given the challenges of learning diverse robotic skills, simulation environments enable collaborative research across global groups and provide standardized benchmarks for evaluating progress. ManiSkill addresses these needs by offering a comprehensive simulation benchmark for robotic learning. It includes extensive 3D assets with significant intra-class topological and geometric diversity, tasks featuring distinct manipulation challenges, native 3D point cloud support, and large-scale demonstration datasets. These features establish a solid

foundation for advancing data-centric approaches in robotics.

Given the simulation environment, the subsequent challenge is collecting large-scale demonstrations for various tasks. This process involves numerous considerations, including object geometry and physical properties, task complexity and duration, and robot morphologies. A straightforward approach is to have humans teleoperate robots to gather demonstrations. However, this method is prohibitively expensive. For instance, in ManiSkill, with thousands of objects and hundreds of scenes, collecting just 10 demonstrations per setup would incur an enormous cost. A more scalable alternative is to employ reinforcement learning (RL) or model predictive control (MPC) to solve tasks and generate demonstrations. However, both approaches rely on high-quality dense rewards, which typically require meticulous design by human experts. To address this limitation, Chapter 2 introduces DrS (Dense Reward learning from Stages), a novel method for learning reusable dense rewards for multi-stage tasks in a data-driven manner. By leveraging the stage structures of tasks, DrS derives high-quality dense rewards from sparse rewards and demonstrations, when available. These learned rewards can be generalized to unseen tasks, significantly reducing the need for human-intensive reward engineering.

By combining reinforcement learning algorithms with the high-quality dense rewards provided by DrS, it becomes feasible to collect demonstrations for many short-horizon tasks. However, a key limitation of RL lies in its difficulty in addressing long-horizon tasks. Chapter 3 presents TR2, a novel approach designed to collect demonstrations for complex long-horizon tasks. TR2 operates in three stages: 1) constructing a paired abstract environment by simplifying geometry and physics; 2) generating abstract trajectories within this simplified environment; and 3) solving the original task by translating abstract trajectories into executable ones through an abstract-to-executable trajectory translator. In the abstract environment, intricate dynamics, such as physical manipulation, are removed, making the generation of abstract trajectories significantly easier. However, this simplification introduces a substantial domain gap between the abstract trajectories and the corresponding executable trajectories, as the former lack low-level details and are not frame-to-frame aligned with the executed trajectories. To address this challenge,

TR2 employs a sequence-to-sequence (seq-to-seq) model, akin to methods used in language translation, to bridge the domain gap. This enables the low-level policy to accurately follow the abstract trajectory, facilitating the execution of complex long-horizon tasks. In this way, a transformer model was developed to translate abstract trajectories into sequences of robot actions. The most significant advantage of this model is its versatility—it can now generate diverse trajectories by simply providing different abstract trajectories as prompts. This capability enables the collection of demonstrations for various long-horizon tasks, greatly enhancing scalability and efficiency.

After collecting demonstrations, the next critical question is how to effectively utilize them. The most straightforward approach is imitation learning. Recent advancements in imitation learning, such as ACT [233] and Diffusion Policy [28], have shown promise. However, the performance of these offline-trained policies remains constrained by the quantity and quality of the demonstrations. Chapter 4 addresses this limitation by exploring ways to enhance offline-trained imitation learning models through online interactions with the environment. It introduces Policy Decorator, a model-agnostic residual policy designed to refine large imitation learning models during online interactions. By employing controlled exploration strategies, Policy Decorator ensures stable and sample-efficient online learning. The evaluation spans eight tasks across two benchmarks—ManiSkill and Adroit—and involves two state-of-the-art imitation learning models: Behavior Transformer and Diffusion Policy. Results demonstrate that Policy Decorator significantly improves the performance of offline-trained policies while preserving the smooth motion characteristics of imitation learning models, thereby avoiding the erratic behaviors typically observed in pure reinforcement learning policies.

In summary, this dissertation discusses the three key components of data-centric robotics: environment, demonstration collection, and policy learning. For the environment, the ManiSkill benchmark is introduced, which is a simulation platform that serves as a foundational source of data. For demonstration collection, two distinct approaches are proposed: one tailored for short-horizon tasks and another designed for long-horizon tasks. Finally, this dissertation presents a

method that integrates off-the-shelf imitation learning techniques with online learning to enhance policy performance.

Chapter 1

ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations

Figure 1.1. A subset of environments in ManiSkill. We currently support 4 different manipulation tasks: OpenCabinetDoor, OpenCabinetDrawer, PushChair, and MoveBucket; each features a large variety of 3D articulated objects to encourage generalizable physical manipulation skill learning.

1.1 Introduction and Related Works

To automate repetitive work and daily chores, robots need to possess human-like manipulation skills. A remarkable feature of human manipulation skill is that, once we have learned

to manipulate a category of objects, we will be able to manipulate even unseen objects of the same category, despite the large topological and geometric variations. Taking swivel chairs as an example, regardless of the existence of an armrest or headrest, the number of wheels, or the shape of the backrest, we are confident of using them immediately. We refer to such ability to interact with unseen objects within a certain category as **generalizable manipulation skills**.

Generalizable manipulation skill learning is at the nexus of vision, learning, and robotics, and poses many interesting research problems. Recently, this field has started to attract much attention across disciplines. For example, reinforcement learning and imitation learning are applied to object grasping and manipulation [86, 180, 105, 169, 100, 8, 111, 144, 229]. On the other hand, [120, 161, 131, 44, 114, 197, 164, 45] can propose novel grasp poses on novel objects based on visual inputs. To further foster synergistic efforts, it is crucial to build a benchmark that backs reproducible research and allows researchers to compare and thoroughly examine different algorithms.

However, building such a benchmark is extremely challenging. To motivate our benchmark proposal, we first analyze key factors that complicate the design of generalizable manipulation skill benchmarks and explain why existing benchmarks are still insufficient. With these motivations in mind, we then introduce the design features of our SAPIEN Manipulation Skills Benchmark (abbreviated as **ManiSkill**).

Key factors that affect our benchmark design.

To guide users and create concentration on algorithm design, four key factors must be considered: 1) manipulation policy structure, 2) diversity of objects and tasks, 3) targeted perception algorithms, and 4) targeted policy algorithms.

1) Manipulation policy structure: Manipulation policies have complex structures that require different levels of simulation support, and we focus on full-physics simulation. Since simulating low-level physics is difficult, many robot simulators only support abstract action space (i.e., manipulation skills already assumed) [93, 212, 155, 215, 214, 176, 56, 43, 212, 56]. It is

convenient to study high-level planning in these benchmarks; however, it becomes impossible to study more challenging scenarios with high-dimensional and complex low-level physics. Some recent benchmarks [17, 195, 235, 204, 226, 81] start to leverage the latest full-physics simulators [200, 36, 172] to support physical manipulation. Despite the quantity of existing environments, most of them lack the ability to benchmark object-level generalizability within categories, and lack inclusion for different methodologies in the community, while we excel in these dimensions, which is explained next.

2) Diversity of objects and tasks: To test object-level generalizability, the benchmark must possess enough intra-class variation of object topology, geometry, and appearance, and we provide such variation. Several benchmarks or environments, including robosuite [235], RLBench [81], and Meta-World [226], feature a wide range of tasks; however, they possess a common problem: lacking object-level variations. Among past works, DoorGym [204] is equipped with the best object-level variations: it is a door opening benchmark with doors procedurally generated from different knob shapes, board sizes, and physical parameters, but it still does not capture some simple real-world variations, such as multiple doors with multiple sizes on cabinets with different shapes. This is in part due to the limitations of procedural modeling. Even though procedural modeling has been used in 3D deep learning [38, 222], it often fails to cover objects with real-world complexity, where crowd-sourced data from Internet users and real-world scans are often preferred (which is our case). Finally, a single type of task like opening doors cannot cover various motion types. For example, pushing swivel chairs requires very different skills from opening doors since it involves controlling under-actuated systems through dual-arm collaboration. Therefore, it is essential to build benchmarks with *both* great asset variations and wide skill coverage.

3) Targeted perception algorithms: Benchmarks need to decide the type and format of sensor data, and we focus on 3D sensor data mounted on robots. Many existing benchmarks, such as *DoorGym*, rely on fixed cameras to capture 2D images; however, this setting greatly limits the tasks a robot can solve. Instead, robot-mounted cameras are common in the real

world to allow much higher flexibility, such as Kinova MOVO, and autonomous driving in general; those cameras are usually designed to capture 3D inputs, especially point clouds. Moreover, tremendous progress has been achieved in building neural networks with 3D input [159, 160, 199, 62, 31, 232, 158], and these 3D networks have demonstrated strong performance (e.g., they give better performance than 2D image networks on autonomous driving datasets [211]). [30, 23, 139, 112, 126, 230] have also adopted 3D deep learning models for perceiving and identifying kinematic structures and object poses for articulated object manipulation. Our benchmark provides users with an ego-centric panoramic camera to capture point cloud / RGB-D inputs. Additionally, we present and evaluate 3D neural network-based policy learning baselines.

4) Targeted policy algorithms: Different policy learning algorithms require different training data and settings, and we provide multiple tracks to advocate for fair comparison. For example, imitation learning [72, 174, 169] and offline RL [55, 98, 106] can learn a policy purely from demonstrations datasets [37, 24], but online RL algorithms [66, 179] require interactions with environments. Therefore, a clear and meaningful split of tracks can encourage researchers with different backgrounds to explore generalizable manipulation skills and let them focus on different aspects of the challenge, e.g., network design, perception, interaction, planning, and control. While oftentimes other benchmarks are limited to a single domain of research and a single modality, our benchmark supports three different tracks for researchers from computer vision, reinforcement learning, and robotics fields.

Our benchmark.

Above we discussed factors of manipulation benchmark design and mentioned the principles behind ManiSkill. Here we introduce the key features of the benchmark. ManiSkill is a large-scale open-source benchmark for physical manipulation skill learning over a diverse set of articulated objects from 3D visual inputs. ManiSkill has four main features: **First**, to support generalizable policy learning, ManiSkill provides objects of high topology and geometry variations, as shown in Fig 1.2. It currently includes a total of 162 objects from 3 object

Figure 1.2. ManiSkill features diverse articulated objects with complex topological and geometric variations, such as different numbers and shapes of doors and/or drawers on different shapes of cabinets. We invested significant effort to process objects from the PartNet-Mobility Dataset and integrate them into our tasks, such as adjusting the size and physical parameters (e.g. friction) so that environments are solvable, along with manual convex decomposition.

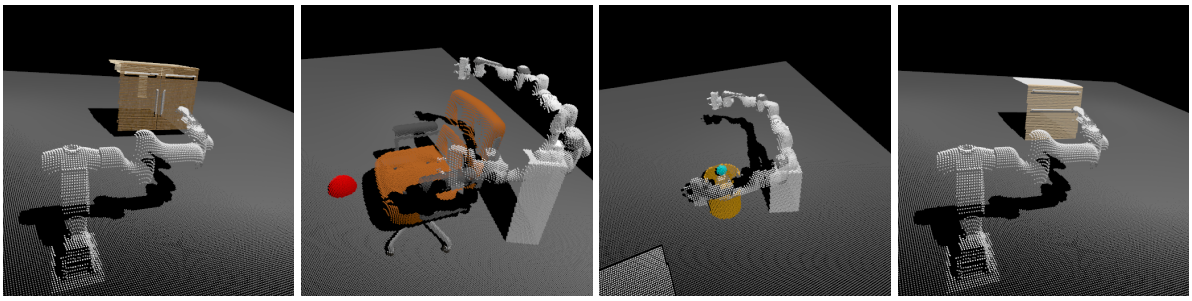

Figure 1.3. Rendered point clouds from our tasks. ManiSkill supports 3D visual inputs which are widely accessible in real environments, allowing various computer vision models to be applied. (For a better view, we show point clouds obtained from cameras mounted in the world frame. In actual tasks, cameras are mounted on the robot head, offering an egocentric view.)

categories (more objects are being added) selected and manually processed from a widely used 3D vision dataset. **Second**, ManiSkill focuses on 4 object-centric manipulation tasks that exemplify household manipulation skills with different types of object motions, thereby posing challenges to distinct aspects of policy design/learning (illustrated in Fig 1.1 and Fig 1.3). As an ongoing effort, we are designing more. **Third**, to facilitate learning-from-demonstration methods, we have collected a large number of successful trajectories (~36,000 trajectories, ~1.5M 3D point cloud / RGB-D frames in total). **Fourth**, the environments feature high-quality data for physical manipulation. We make significant efforts to select, fix, and re-model the original PartNet-Mobility data [216, 129, 25], as well as design the reward generation rules so that the manipulation task of each object can be solved by an RL algorithm.

A major challenge to build our benchmark is to collect demonstrations. Some tasks

are tricky (e.g., swivel chair pushing requires dual-arm coordination), and it is difficult and unscalable to manually control the robots to collect large-scale demos. It is also unclear whether traditional motion planning pipelines can solve all tasks. Thankfully, reinforcement learning does work for individual objects, allowing for a divide-and-conquer approach to create high-quality demonstrations. With a meticulous effort on designing a shared reward template to automatically generate reward functions for all object instances of each task and executing an RL agent for each object instance, we are able to collect a large number of successful trajectories. This RL plus divide-and-conquer approach is very scalable with respect to the number of object instances within a task, and we leave cross-task RL reward design for future work. It is worth noting that, *we do NOT target at providing a GENERIC learning-from-demonstrations benchmark* that compares methods from all dimensions. Instead, we compare the ability of different algorithms to utilize our demonstrations to solve manipulation tasks.

Another important feature of ManiSkill is that it is completely free and built on an entirely open-source stack. Other common **physical** manipulation environments, including robosuite[235], DoorGym[204], MetaWorld[226], and RLBench[81], depend on commercial software.

To summarize, here are the key contributions of ManiSkill Benchmark.

- The topology and geometry variation of our data allows our benchmark to compare object-level generalizability of different physical manipulation algorithms. Our data is high-quality, and every object is verified to support RL.
- The manipulation tasks we design target distinct challenges of manipulation skills (by motion types, e.g. revolute and prismatic joint constraints, or by skill properties, e.g. requirement of dual-arm collaboration).
- ManiSkill provides a large-scale demonstrations dataset with ~36,000 trajectories and ~1.5M point cloud/RGB-D frames to facilitate learning-from-demonstrations approaches.

The demonstrations are collected by a scalable RL approach with dense rewards generated by a shared reward template within each task.

- We provide several 3D deep learning-based policy baselines.

1.2 ManiSkill Benchmark

The goal of building the ManiSkill benchmark can be best described as facilitating *learning generalizable manipulation skills from 3D visual inputs with demonstrations*. "*Manipulation*" involves low-level physical interactions and dynamics simulation between robot agents and objects; "*skills*" refer to short-horizon physics-rich manipulation tasks, which can be viewed as basic building blocks of more complicated policies; "*3D visual inputs*" are egocentric point cloud and/or RGB-D observations captured by a panoramic camera mounted on a robot; "*demonstrations*" are trajectories that solve tasks successfully to facilitate learning-from-demonstrations approaches.

In this section, we will describe the components of the ManiSkill benchmark in detail, including basic terminologies and setup, task design, demonstration trajectory collection, training-evaluation protocol, and asset postprocessing with verification.

Table 1.1. Dataset statistics for ManiSkill. For OpenCabinetDoor and OpenCabinetDrawer, numbers outside of the parenthesis indicate the number of unique cabinets, where each cabinet may have more than one door/drawer. Numbers in the parenthesis indicate the total number of doors/drawers. * The DoF in the table indicates the DoF involved in solving a task. For OpenCabinetDoor and OpenCabinetDrawer, an agent only needs to open one designated door/drawer. For PushChair and MoveBucket, 6 extra DoF are included since chairs and buckets can move freely in 3D space.

Task	# objects			Dual-arm Collaboration?	Solvable by Motion Planning?	DoF*
	All	Train	Test			
OpenCabinetDoor	52(82)	42(66)	10(16)	No	Easy	1
OpenCabinetDrawer	35(70)	25(49)	10(21)	No	Easy	1
PushChair	36	26	10	Yes	Hard	∼15-25
MoveBucket	39	29	10	Yes	Medium	7

1.2.1 Basic Terminologies and Setup

In ManiSkill, a **task** or a **skill** $\mathscr{T} = \{T_{o,l} : o \in \mathscr{O}, l \in \mathscr{L}_o\}$ consists of finite-horizon POMDPs (Partially Observable Markov Decision Processes) defined over a set of objects $\mathscr{O}$ of the same category (e.g., chairs) and a set of environment parameters $\mathscr{L}_o$ associated with an object $o \in \mathscr{O}$ (e.g. friction coefficients of joints on a chair). An **environment** is a set of POMDPs $\mathscr{E}_o = \{T_{o,l} : l \in \mathscr{L}_o\}$ defined over a single object o and its corresponding parameters. Each $T_{o,l}$ is a specific instance of an environment, represented by a tuple of sets (S, A, P, R, O). Here, $s \in S$ is an environment state that consists of *robot states* (e.g. joint angles of the robot) and *object states* (e.g. object pose and the joint angles); $a \in A$ is an action that can be applied to a robot (e.g. target joint velocity of a velocity controller); $P(s'|s,a)$ is the physical dynamics; R is a binary variable that indicates if the task is successfully solved; $O(o|s)$ is a function which generates observations from an environment state, and it supports three modes in ManiSkill: `state`, `pointcloud`, and `rgbd`. In `state` mode, the observation is identical to s. In `pointcloud` and `rgbd` modes, the *object states* in s are replaced by the corresponding point cloud / RGB-D visual observations captured from a panoramic camera mounted on a robot. `state` mode is not suitable for studying generalizability, as *object states* are not available in realistic setups, where information such as object pose has to be estimated based on some forms of visual inputs that are universally obtainable (e.g. point clouds and RGB-D images).

For each task, objects are partitioned into training objects $\mathscr{O}_{train}$ and test objects $\mathscr{O}_{test}$, and environments are divided into training environments $\{T_{o,l}|o \in \mathscr{O}_{train}\}$ and test environments $\{T_{o,l}|o \in \mathscr{O}_{test}\}$. For each training environment, successful demonstration trajectories are provided to facilitate learning-from-demonstrations approaches.

We define **object-level generalizable manipulation skill** as a manipulation skill that can generalize to unseen test objects after learning on training objects where the training and test objects are from the same category. Some notable challenges of our tasks come from partial observations (i.e. point clouds / RGB-D images only covering a portion of an object), robot

arms occluding parts of an object, and complex shape understanding over objects with diverse topological and geometric properties.

1.2.2 Tasks with Diverse Motions and Skills

Object manipulation skills are usually associated with certain types of desired motions of target objects, e.g., rotation around an axis. Thus, the tasks in ManiSkill are designed to cover different types of object motions. We choose four common types of motion constraints: revolute joint constraint, prismatic joint constraint, planar motion constraint, and no constraints, and build four tasks to exemplify each of these motion types. In addition, different tasks also feature different properties of manipulation, such as dual-arm collaboration and solvability by motion planning. Statistics for our tasks are summarized in Table 1.1. Descriptions for our tasks are stated below.

OpenCabinetDoor exemplifies motions constrained by a revolute joint. In this task, a single-arm robot is required to open a designated door on a cabinet. The door motion is constrained by a revolute joint attached to the cabinet body. This task is relatively easy to solve by traditional motion planning and control pipelines, so it is suitable for comparison between learning-based methods and motion planning-based methods.

OpenCabinetDrawer exemplifies motions constrained by a prismatic joint. This task is similar to OpenCabinetDoor, but the robot needs to open a target drawer on a cabinet. The drawer motion is constrained by a prismatic joint attached to the cabinet body.

PushChair exemplifies motions constrained on a plane through wheel-ground contact. A dual-arm robot needs to push a swivel chair to a target location on the ground and prevent it from falling over. PushChair exemplifies the ability to manipulate complex underactuated systems, as swivel chairs generally have many joints, resulting in complex dynamics. Therefore, it is difficult to solve PushChair by motion planning and favors learning-based methods.

MoveBucket exemplifies motions without constraints. In this task, a dual-arm robot needs to move a bucket with a ball inside and lift it onto a platform. There are no constraints

on the motions of the bucket. However, this task is still very challenging because: 1) it heavily relies on two-arm coordination as the robot needs to lift the bucket; 2) the center of mass of the bucket-ball system is consistently changing, making balancing difficult.

Note that all environments in ManiSkill are verified to be solvable, i.e., for each object, we guarantee that there is a way to manipulate it to solve the corresponding task. This is done by generating successful trajectories in each environment (details in Sec 1.2.4). Instead of creating lots of tasks but leaving the solvability problems to users, our tasks are constructed with appropriate difficulty and verified solvability.

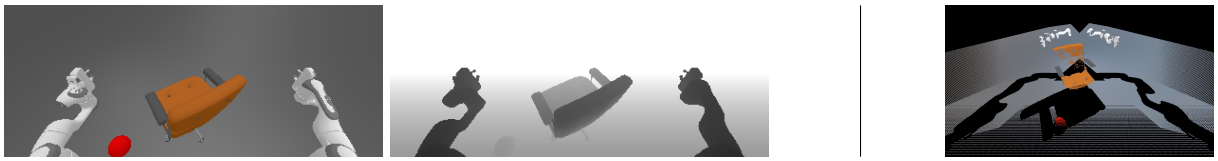

Figure 1.4. RGB-D (RGB/Depth) and point cloud observations in ManiSkill. Left two images: RGB-D image from *one* of the three cameras mounted on the robot. The three cameras together provide an ego-centric panoramic view. Right image: visualization of the fused point cloud from *all* three cameras. The center of the robot body cannot be seen since the captured point cloud comes from an ego-centric view. Parts of the chair are occluded by itself (as cameras are mounted on the robot).

1.2.3 Robots, Actions, Visual Observations, and Rewards

All the tasks in ManiSkill use similar robots, which are composed of three parts: a moving platform, Sciurus robot body, and one or two Franka Panda arm(s). The moving platform can move and rotate on the ground plane, and its height is adjustable. The robot body is fixed on top of the platform, providing support for the arms. Depending on the task, one or two robot arm(s) are connected to the robot body. There are 22 joints in a dual-arm robot and 13 in a single-arm robot. To match realistic robotics setups, we use PID controllers to control the joints of robots. The action space corresponds to the normalized target values of all controllers. In addition to joint space control, ManiSkill supports operational space control, i.e., directly controlling the end-effectors in the Cartesian space.

As mentioned in Sec 1.2.1, ManiSkill supports three observation modes: state, pointcloud, and rgbd, where the latter two modes are suitable for studying object-level generalizability. The RGB-D and point cloud observations are captured from three cameras mounted on the robot to provide an ego-centric panoramic view, resembling common real-world robotics setups. The three cameras are 120° apart from each other, and the resolution of each camera is 400×160. The observations from all cameras are combined to form a final panoramic observation. Visualizations of RGB-D / point cloud observations are shown in Fig 1.4. In addition, we provide some task-relevant segmentation masks for both RGB-D and point cloud observations.

ManiSkill supports two kinds of rewards: `sparse` and `dense`. A sparse reward is a binary signal that is equivalent to a task-specific success condition. Learning with sparse rewards is very difficult. To alleviate such difficulty, we carefully designed well-shaped dense reward functions for each task. The dense rewards are also used in demonstration collection, which will be elaborated in Sec 1.2.4.

1.2.4 RL-Based Demo Collection and MPC-Assisted Reward Template Design

Our interactive environment naturally supports methods such as reinforcement learning or classical robotics pipelines. However, to build an algorithm with object-level generalizability either by RL or by designing rules, it is probably prohibitively complex and resource-demanding for many researchers interested in manipulation learning research (e.g., vision researchers might be primarily interested in the perception module). We observe that many learning-from-demonstrations algorithms (e.g., behavior cloning) are much easier to start with and require less resources.

To serve more researchers, ManiSkill provides a public demonstration dataset with a total of ~36,000 successful trajectories and ~1.5M frames (300 trajectories for each training object in each task). The demonstrations are provided in the format of internal environment states to save storage space, and users can render the corresponding point cloud / RGB-D frames using the

provided scripts.

In order to construct this large-scale demonstration dataset, we need a **scalable** pipeline that can produce plentiful demonstrations automatically. Compared to other existing approaches (e.g. human annotation by teleoperation, motion planning), we adopted a reinforcement learning (RL)-based pipeline, which requires significantly less human effort and can generate an arbitrary number of demonstrations at scale. Though it is common to collect demonstrations by RL [49, 116, 107, 72, 15, 89, 83, 237], directly training a single RL agent to collect demonstrations for all environments in ManiSkill is challenging because of the large number of different objects and the difficulty of our manipulation tasks. To collect demonstrations at scale, we design an effective pipeline as follows.

Our pipeline contains two stages. In the first stage, we need to design and verify dense rewards for RL agents. For each task, we design a shared reward template based on the skill definitions of this task with humans prior. *Note that this reward template is shared across all the environments (objects) in a task instead of manually designed for each object.* In order to *quickly verify* the reward template (as our tasks are complicated and solving by RL takes hours), we use Model-Predictive Control (MPC) via Cross-Entropy Method (CEM), which can be efficiently parallelized to find a trajectory within 15 minutes (if successful) from *one single initial state* using 20 CPUs. While MPC is an efficient tool to verify our reward template, it is not suitable for generating our demonstration dataset, which should contain diverse and randomized initial states. This is because MPC has to be retrained independently each time to find a trajectory from each of the 300 initial states for each training object of each task, rendering it unscalable. Therefore, in the second stage, we train model-free RL agents to collect demonstrations. We also found that training one single RL agent on many environments (objects) of a task is very challenging, but training an agent to solve a single specific environment is feasible and well-studied. Therefore, we collect demonstrations in a divide-and-conquer way: for each environment, we train a SAC [66] agent and generate successful demonstrations.

1.2.5 Multi-Track Training-Evaluation Protocol

As described in Sec 1.2.1, agents are trained on the training environments with their corresponding demonstrations and evaluated on the test environments for object-level generalizability, under `pointcloud` or `rgbd` mode. Moreover, the ManiSkill benchmark aims to encourage interdisciplinary insights from computer vision, reinforcement learning, and robotics to advance generalizable physical object manipulation. To this end, we have developed our benchmark with 3 different tracks:

No Interactions: this track requires solutions to only use our provided demonstration trajectories during training. No interactions (i.e. additional trajectory collection, online training, etc.) are allowed. For this track, solutions may choose to adopt a simple but effective supervised learning algorithm — matching the predicted action with the demonstration action given visual observation (i.e. behavior cloning). Therefore, this track encourages researchers to explore 3D computer vision network architectures for generalizable shape understanding over complex topologies and geometries.

No External Annotations: this track allows online model fine-tuning over training environments on top of No Interactions track. However, the solution must not contain new annotations (e.g. new articulated objects from other datasets). This track encourages researchers to explore online training algorithms, such as reinforcement learning with online data collection.

No Restrictions: this track allows solutions to adopt any approach during training, such as labeling new data and creating new environments. Researchers are also allowed to use manually designed control and motion planning rules, along with other approaches from traditional robotics.

The benchmark evaluation metric is the *mean success rate* on a predetermined set of test environment instances. For each task, we have defined the success condition, which is automatically reported in the evaluation script provided by us. Each track should be benchmarked separately.

1.2.6 Asset Selection, Re-Modeling, Postprocessing, and Verification

While the PartNet-Mobility dataset (from SAPIEN [216]) provides a repository of articulated object models, the original dataset can only provide full support for vision tasks such as joint pose estimation. Therefore, we make significant efforts to select, fix, and verify the models.

First, the PartNet-Mobility dataset is not free of annotation errors. For example, some door shafts are annotated on the same side as the door handles. While it does not affect the simulation, such models are unnatural and not good candidates to test policy generalizability. Thus, we first render all assets and manually exclude the ones with annotation errors.

Moreover, fast simulation requires a convex decomposition of 3D assets. However, the automatic algorithm used in the original SAPIEN paper, VHACD [121], cannot handle all cases well. For example, VHACD can introduce unexpected artifacts, such as dents on smooth surfaces, which agents can take advantage of. To fix the errors, we identify problematic models by inspection and use Blender 's [34] shape editing function to manually decompose the objects.

Even with all the efforts above, some models can still present unexpected behaviors. For example, certain cabinet drawers may be stuck due to inaccurate overlapping between collision shapes. Therefore, we also verify each object by putting them in the simulator and learning a policy following Sec 1.2.4. We fix issues if we cannot learn a policy to achieve the task. We repeat until all models can yield a successful policy by MPC.

1.3 Baseline Architectures, Algorithms, and Experiments

Learning object-level generalizable manipulation skills through 3D visual inputs and learning-from-demonstrations algorithms has been underexplored. Therefore, we designed several baselines and open-sourced their implementations here to encourage future explorations in the field.

We adopted `pointcloud` observation mode and designed point cloud-based vision archi-

tectures as our feature extractor since previous work [211] has achieved significant performance improvements by using point clouds instead of RGB-D images. Point cloud features include position, RGB, and segmentation masks, and we concatenate the *robot state* to the features of each point. Intuitively, this allows the extracted feature to not only contain geometric information about objects but also contain the relation between the robot and each individual object, such as the closest point to the robot, which is very difficult to learn without such concatenation. In addition, we downsample the point cloud data to increase training speed and reduce the memory footprint.

The first point cloud-based architecture uses one single PointNet [159], a very popular 3D deep learning backbone, to extract a global feature for the entire point cloud, which is fed into the final MLP. The second architecture uses different PointNets to process points belonging to different segmentation masks. The global features from the PointNets are then fed into a Transformer [206], after which a final attention pooling layer extracts the final representations and feeds into the final MLP. We designed and benchmarked this architecture since it allows the model to capture the relation between different objects and possibly provides better performance. While there is great room to improve, we believe that these architectures could provide good starting points for many researchers.

For learning-from-demonstrations algorithms on top-of-point cloud architectures, we benchmark two approaches - Imitation Learning (IL) and Offline/Batch Reinforcement Learning (Offline/Batch RL). For imitation learning, we choose a simple and widely adopted algorithm: behavior cloning (BC) - directly matching predicted and ground truth actions through minimizing L_2 distance. For offline RL, we benchmark Batch-Constrained Q-Learning (BCQ) [55] and Twin-Delayed DDPG[53] with Behavior Cloning (TD3+BC) [52]. We follow their original implementations and tune the hyperparameters.

Table 1.2. The average success rates of different agents on **one single environment** (fixed object instance) of OpenCabinetDrawer with different numbers of demonstration trajectories. The average success rates are calculated over 100 evaluation trajectories. While network architectures and algorithms play an important role in the performance, learning manipulation skills from demonstrations is challenging without a large number of trajectories, even in one single environment.

#Demo Trajectories	10	30	100	300	1000
#Gradient Steps	2000	4000	10000	20000	40000
PointNet, BC	0.13	0.23	0.37	0.68	0.76
PointNet + Transformer, BC	0.16	0.35	0.51	0.85	0.90
PointNet + Transformer, BCQ	0.02	0.05	0.23	0.45	0.55
PointNet + Transformer, TD3+BC	0.03	0.13	0.22	0.31	0.57

1.3.1 Single Environment Results

As a glimpse into the difficulty of learning manipulation skills from demonstrations in our benchmark, we first present results with an increasing number of demonstration trajectories on **one single environment** of OpenCabinetDrawer in Table 1.2. We observe that the success rate gradually increases as the number of demonstration trajectories increases, which shows the agents can indeed benefit from more demonstrations. We also observe that inductive bias in network architecture plays an important role in the performance, as PointNet + Transformer is more sample efficient than PointNet. Interestingly, we did not find offline RL algorithms to outperform BC. We conjecture that this is because the provided demonstrations are all successful ones, so an agent is able to learn a good policy through BC. In addition, our robot's high degree of freedom and the difficulty of the task itself pose a challenge to offline RL algorithms. It is worth noting that our experiment results should not discourage benchmark users from including failure trajectories and find better usage of offline RL methods, especially those interested in the No External Annotations track described in Sec 1.2.5.

Table 1.3. Mean and standard deviation of *average success rates* on **training and test environments** of each task over 5 different runs, under the point cloud observation. Models are trained with our demonstration dataset, with 300 demonstration trajectories per training environment. For each task, the average test success rates are calculated over the 10 test environments and 50 evaluation trajectories per environment. Obtaining one single agent capable of learning manipulation skills across multiple objects and generalizing the learned skills to novel objects is challenging.

Algorithm	BC				BCQ		TD3+BC	
Architecture	PointNet		PointNet + Transformer		PointNet + Transformer		PointNet + Transformer	
Split	Training	Test	Training	Test	Training	Test	Training	Test
OpenCabinetDoor	0.18±0.02	0.04±0.03	0.30±0.06	0.11±0.02	0.16±0.02	0.04±0.02	0.13±0.03	0.04±0.02
OpenCabinetDrawer	0.24±0.03	0.11±0.03	0.37±0.06	0.12±0.02	0.22±0.04	0.11±0.03	0.18±0.02	0.10±0.02
PushChair	0.11±0.02	0.09±0.02	0.18±0.02	0.08±0.01	0.11±0.01	0.08±0.01	0.12±0.02	0.08±0.01
MoveBucket	0.03±0.01	0.02±0.01	0.15±0.01	0.08±0.01	0.08±0.01	0.06±0.01	0.05±0.01	0.03±0.01

1.3.2 Object-Level Generalization Results

We now present results on **object-level generalization**. We train each model for 150k gradient steps. This takes about 5 hours for BC, 35 hours for BCQ, and 9 hours for TD3+BC using the PointNet + Transformer architecture on one NVIDIA RTX 2080Ti GPU. As shown in Table 1.3, even with our best agent (BC PointNet + Transformer), the overall success rates in both training and test environments are low. We also observe that the training accuracy over object variations is significantly lower than the training accuracy on one single environment (in Table 1.2). The results suggest that existing works on 3D deep learning and learning-from-demonstrations algorithms might have been insufficient yet to achieve good performance when trained for physical manipulation skills over diverse object geometries and tested for object-level generalization. Therefore, we believe there is a large space to improve, and our benchmark poses interesting and challenging problems for the community.

1.4 Conclusion and Limitations

In this work, we propose ManiSkill, an articulated benchmark for generalizable physical object manipulation from 3D visual inputs with diverse object geometries and large-scale

demonstrations. We expect ManiSkill would encourage the community to look into object-level generalizability of manipulation skills, specifically by combining cutting-edge research of 3D computer vision, reinforcement learning, and robotics.

Our benchmark is limited in the following aspects: 1) Currently, we provide 162 articulated objects in total. We plan to process more objects from the PartNet-Mobility dataset [216] and add them to our ManiSkill assets; 2) While the four tasks currently provided in ManiSkill exemplify distinct manipulation challenges, they do not comprehensively cover manipulation skills in household environments. We plan to add more tasks among the same skill properties (e.g, pouring water from one bucket to another bucket through two-arm coordination); 3) We have not conducted sim-to-real experiments yet, and this will be a future direction of ManiSkill.

1.5 Acknowledgment

We thank Qualcomm for sponsoring the associated challenge, Sergey Levine and Ashvin Nair for insightful discussions during the whole development process, Yuzhe Qin for the suggestions on building robots, Jiayuan Gu for providing technical support on SAPIEN, and Rui Chen, Songfang Han, Wei Jiang for testing our system.

This chapter, in full, is taken from "ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations" in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track 2021 by Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, Hao Su. The dissertation author was a primary investigator and author of this paper.

Chapter 2

DrS: Learning Reusable Dense Rewards for Multi-Stage Tasks

2.1 Introduction

The success of many reinforcement learning (RL) techniques heavily relies on dense reward functions [76, 150], which are often tricky to design by humans due to heavy domain expertise requirements and tedious trials and errors. In contrast, sparse rewards, such as a binary task completion signal, are significantly easier to obtain (often directly from the environment). For instance, in pick-and-place tasks, the sparse reward could simply be defined as the object being placed at the goal location. Nonetheless, sparse rewards also introduce challenges (e.g., exploration) for RL algorithms [148, 22, 41]. Therefore, a crucial question arises: *can we learn dense reward functions in a data-driven manner?*

Ideally, the learned reward will be **reused** to efficiently solve new tasks that share similar success conditions with the task used to learn the reward. For example, in pick-and-place tasks, different objects may need to be manipulated with varying dynamics, action spaces, and even robot morphologies. For clarity, we refer to each variant as a *task* and the set of all possible pick-and-place tasks as a *task family*. Importantly, the reward function, which captures approaching, grasping, and moving the object toward the goal position, can potentially be transferred within this task family. This observation motivates us to explore the concept of *reusable rewards*, which can be learned as a function from some tasks and reused in unseen tasks. While existing literature

in RL primarily focuses on the reusability (generalizability) of policies, we argue that rewards can pose greater flexibility for reuse across tasks. For example, it is nearly impossible to directly transfer a policy operating a two-finger gripper for pick-and-place to a three-finger gripper due to action space misalignment, but a reward inducing the approach-grasp-move workflow may apply for both types of grippers.

However, many existing works on reward learning do not emphasize reward reuse for new tasks. The field of learning a reward function from demonstrations is known as inverse RL in the literature [141, 1, 238]. More recently, adversarial imitation learning (AIL) approaches have been proposed [73, 95, 50, 59] and gained popularity. Following the paradigm of GANs [61], AIL approaches employ a policy network to generate trajectories and train a discriminator to distinguish between agent trajectories from demonstration ones. By using the discriminator score as rewards, [73] shows that a policy can be trained to imitate the demonstrations. Unfortunately, such rewards are *not reusable* across tasks – at convergence, the discriminator outputs $\frac{1}{2}$ for both the agent trajectories and the demonstrations, as discussed in [61, 50], making it unable to learn useful information for solving new tasks.

In contrast to AIL, we propose a novel approach for learning *reusable rewards*. Our approach involves incorporating sparse rewards as a supervision signal in lieu of the original signal used for classifying demonstration and agent trajectories. Specifically, we train a discriminator to classify *success trajectories* and *failure trajectories* based on the binary sparse reward. Please refer to Fig. 2.2 (a)(b) for an illustrative depiction. Our formulation assigns higher rewards to transitions in success trajectories and lower rewards to transitions within failure trajectories, which is consistent throughout the entire training process. As a result, the reward will be reusable once the training is completed. Expert demonstrations can be included as success trajectories in our approach, though they are not mandatory. We only require the availability of a sparse reward, which is a relatively weak requirement as it is often an inherent component of the task definition.

Our approach can be extended to leverage the inherent structure of *multi-stage tasks* and derive stronger dense rewards. Many tasks naturally exhibit multi-stage structures, and it is

relatively easy to assign a binary indicator on whether the agent has entered a stage. For example, in the "Open Cabinet Door" task depicted in Fig. 2.1, there are three stages: 1) approach the door handle, 2) grasp the handle and pull the door, and 3) release the handle and keeping it steady. If the agent is grasping the handle of the door but the door has not been opened enough, then we can simply use a corresponding binary indicator asserting that the agent is in the 2nd stage. By utilizing these stage indicators, we can learn a dense reward for each stage and combine them into a more structured reward. Since the horizon for each stage is shorter than that of the entire task, learning a high-quality dense reward becomes more feasible. Furthermore, this approach provides flexibility in incorporating extra information beyond the final success signal. We dub our approach as **DrS** (**D**ense **r**eward learning from **S**tages).

Our approach exhibits strong performance on challenging tasks. To assess the reusability of the rewards learned by our approach, we employ the ManiSkill benchmark [132, 64], which offers a large number of task variants within each task family.We evaluate our approach on three task families: Pick-and-Place, Open Cabinet Door, and Turn Faucet, including 1000+ task variants. Each task variant involves manipulating a different object and requires precise low-level physical control, thereby highlighting the need for a good dense reward. Our results demonstrate that the learned rewards can be reused across tasks, leading to improved performance and sample efficiency of RL algorithms compared to using sparse rewards. In certain tasks, the learned rewards even achieve performance comparable to those attained by human-engineered reward functions.

Moreover, our approach **drastically reduces the human effort** needed for reward engineering. For instance, while the human-engineered reward for "Open Cabinet Door" involves *over 100 lines of code, 10 candidate terms, and tons of "magic" parameters*, our approach only requires *two boolean functions* as stage indicators: if the robot has grasped the handle and if the door is open enough.

Our contributions can be summarized as follows:

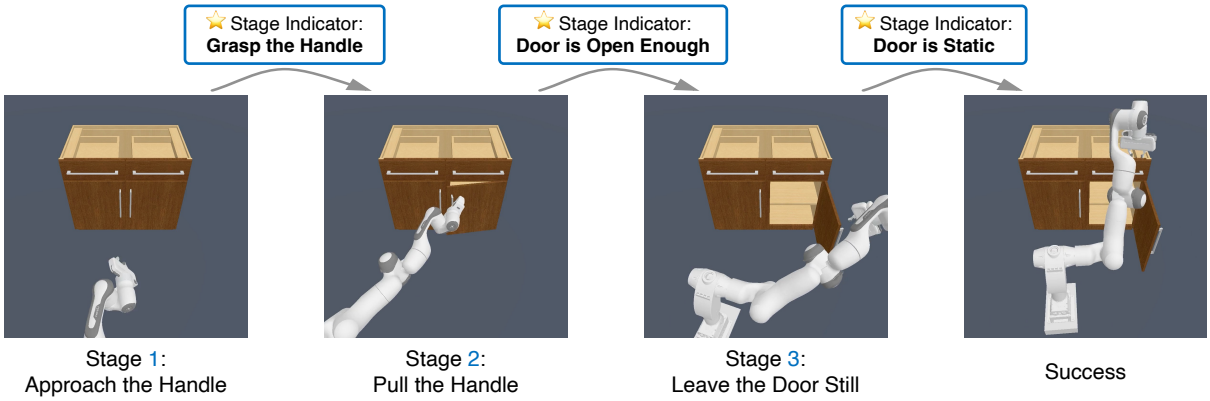

Figure 2.1. An illustration of stage indicators in an OpenCabinetDoor task, which can be naturally divided into three stages plus a success state. A stage indicator is a binary function representing whether the current state is in a certain stage, and it can be simply defined by some boolean functions.

- We propose **DrS** (**D**ense **r**eward learning from **S**tages), a novel approach for learning reusable dense rewards for multi-stage tasks, effectively reducing human efforts in reward engineering.
- Extensive experiments on 1,000+ task variants from three task families showcase the effectiveness of our approach in generating high-quality and reusable dense rewards.

2.2 Related Works

Learning Reward from Demo (Offline) Designing rewards is challenging due to domain knowledge requirements, so approaches to learning rewards from data have gained attention. Some methods adopt classification-based rewards, i.e., training a reward by classifying goals [187, 88, 39] or demonstration trajectories [239]. Other methods [227, 11] use the distance to goal as a reward function, where the distance is usually computed in a learned embedding space, but these methods usually require that the goal never changes in a task. These rewards are only trained on offline datasets, hence they can *easily be exploited* by an RL agent, i.e., an RL can enter a state that is not in the dataset and get a wrong reward signal, as studied in [209, 219].

Learning Reward from Demo (Online) The above issue can be addressed by allowing

agents to verify the reward in the environment, and inverse reinforcement learning (IRL) is the prominent paradigm. IRL aims to recover a reward function given expert demonstrations. Traditional IRL methods [141, 1, 238, 171] often require multiple iterations of Markov Decision Process solvers [157], resulting in poor sample efficiency. In recent years, adversarial imitation learning (AIL) approaches are proposed [73, 95, 50, 59, 116]. They operate similarly to generative adversarial networks (GANs) [61], in which a generator (the policy) is trained to maximize the confusion of a discriminator, and the discriminator (serves the role of rewards) is trained to classify the agent trajectories and demonstrations. However, such rewards are *not reusable* as we discussed in the introduction - classifying agent trajectories and demonstrations is impossible at convergence. In contrast, our approach gets rid of this issue by classifying the success/failure trajectories instead of expert/agent trajectories.

Learning Reward from Human Feedback Recent studies [32, 77, 79] infer the reward through human preference queries on trajectories or explicitly asking for trajectory rankings [20]. Another line of works [51, 186] involves humans specifying desired outcomes or goals to learn rewards. However, in these methods, the rewards only distinguish goal from non-goal states, offering relatively weak incentives to agents at the beginning of an episode, especially in long-horizon tasks. In contrast, our approach classifies all the states in the trajectories, providing strong guidance throughout the entire episode.

Reward Shaping Reward shaping methods aim to densify sparse rewards. Earlier works [140] study the forms of shaped rewards that induce the same optimal policy as the ground-truth reward. Recently, some works [202, 213] have shaped the rewards as the distance to the goal, similar to some offline reward learning methods mentioned above. Another idea [125] involves shaping delayed reward by ranking trajectories based on a fine-grained preference oracle. In contrast to these reward shaping approaches, our method leverages demonstrations, which are available in many real-world problems [192, 37]. This not only boosts the reward learning process but also reduces the additional domain knowledge required by these methods.

Task Decomposition The decomposition of tasks into stages/sub-tasks has been explored

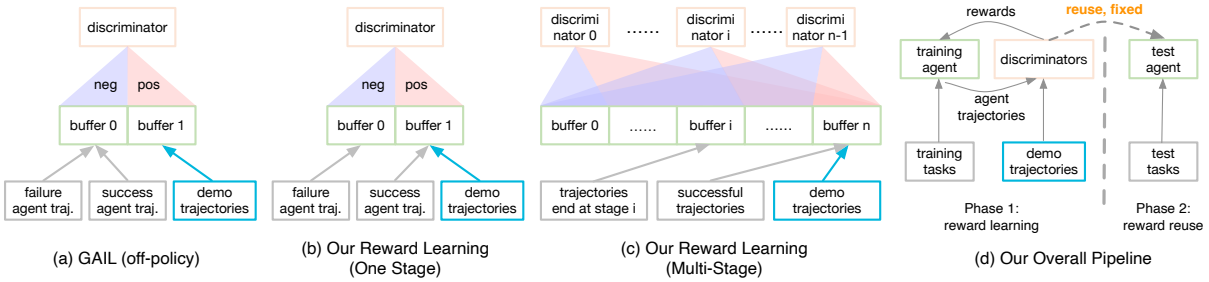

Figure 2.2. a) GAIL's discriminator aims to distinguish agent trajectories from demonstrations. b) In single-stage tasks, the discriminator in our approach aims to distinguish success trajectories from failure ones. c) In multi-stage tasks, our approach train a separate discriminator for each stage. The discriminator for stage k aims to distinguish trajectories that reach beyond stage k from those that only reach up to stage k. d) Overall, our approach has 2 phases: **reward learning** and **reward reuse**.

in various domains. Hierarchical RL approaches [48, 134, 109] break down policies into sub-policies to solve specific sub-tasks. Skill chaining methods [103, 63, 104] focus on solving long-horizon tasks by combining multiple short-horizon policies or skills. Recently, language models have also been utilized to break the whole task into sub-tasks [6]. In contrast to these approaches that utilize stage structures in policy space, our work explores an orthogonal direction by designing rewards with stage structures.

2.3 Problem Setup

In this work, we adopt the Markov Decision Process (MDP) $\mathscr{M} := \langle S, A, T, R, \gamma \rangle$ as the theoretical framework, where R is a reward function that defines the goal or purpose of a task. Specifically, we focus on tasks with *sparse rewards*. In this context, "sparse reward" denotes a binary reward function that gives a value of 1 upon successful task completion and 0 otherwise:

$$R_{sparse}(s) = \begin{cases} 1 & \text{task is completed by reaching one of the success states } s \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Our objective is to learn a dense reward function from a set of training tasks, with the

intention of reusing it for unseen test tasks. Specifically, we aim to successfully train RL agents from scratch on the test tasks using the learned rewards. The desired outcome is to enhance the efficiency of RL training, surpassing the performance achieved by sparse rewards.

We assume that both the training and test tasks are in the same *task family*. A task family refers to a set of task variants that share the same success criteria, but may differ in terms of assets, initial states, transition functions, and other factors. For instance, the task family of object grasping includes tasks such as "Alice robot grasps an apple" and "Bob robot grasps a pen." The key point is that tasks within the same task family share a common underlying sparse reward.

Additionally, we posit that the task can be segmented into multiple stages, and the agent has access to several *stage indicators* obtained from the environment. A stage indicator is a binary function that indicates whether the current state corresponds to a specific stage of the task. An example of stage indicators is in Fig. 2.1. This assumption is quite general as many long-term tasks have multi-stage structures, and determining the current stage of the task is not hard in many cases. By utilizing these stage indicators, it becomes possible to construct a reward that is slightly denser than the binary sparse reward, which we refer to as a *semi-sparse* reward, and it serves as a strong baseline:

$$R_{semi-sparse}(s) = k, \text{ when state } s \text{ is at stage } k \tag{2.2}$$

We aim to design an approach that learns a dense reward based on the stage indicators. When expert demonstration trajectories are available, they can also be incorporated to boost the learning process.

Note that the stage indicators are only required during RL training, but *not required when deploying the policy to the real world*. Training RL agents directly in the real world is often impractical due to cost and safety issues. Instead, a more common practice is to train the agent in simulators and then transfer/deploy it to the real world. While obtaining the stage indicators in simulators is fairly easy, it is also possible to obtain them in the real world by

various techniques (robot proprioception, tactile sensors [115, 124], visual detection/tracking [87, 88], large vision-language models [39], etc.).

2.4 DrS: Dense reward learning from Stages

Dense rewards are often tricky to design by humans, so we aim to learn a reusable dense reward function from stage indicators in multi-stage tasks and demonstrations when available. Overall, our approach has two phases, as shown in Fig. 2.2 (d):

- **Reward Learning Phase**: learn the dense reward function using training tasks.
- **Reward Reuse Phase**: reuse the learned dense reward to train new RL agents in test tasks.

Since the reward reuse phase is just a regular RL training process, we only discuss the reward learning phase in this section. We first explain how our approach learns a dense reward in one-stage tasks (Sec. 2.4.1). Then, we extend this approach to multi-stage tasks (Sec. 2.4.2).

2.4.1 Reward Learning on One-Stage Tasks

In line with previous work [209, 51], we employ a classification-based dense reward. We train a classifier to distinguish between good and bad trajectories, utilizing the learned classifier as dense reward. Essentially, states resembling those in good trajectories receive higher rewards, while states resembling bad trajectories receive lower rewards. While previous Adversarial Imitation Learning (AIL) methods [73, 95] used discriminators as classifiers/rewards to distinguish between agent and demonstration trajectories, these discriminators cannot be directly *reused* as rewards to train new RL agents. As the policy improves, the agent trajectories (negative data) and the demonstrations (positive data) can become nearly identical. Therefore, at convergence, the discriminator output for both agent trajectories and demonstrations tends to approach $\frac{1}{2}$, as observed in GANs [61] (also noted by [50, 219]). This makes it unable to learn useful info for solving new tasks.

Our approach introduces a simple modification to existing AIL methods to ensure that the discriminator continues to learn meaningful information even at convergence. The key issue previously mentioned arises from the diminishing gap between agent and demonstration trajectories over time, making it challenging to differentiate between positive and negative data. To address this, we propose training the discriminator to distinguish between success and failure trajectories instead of agent and demonstration trajectories. By defining success and failure trajectories based on the sparse reward signal from the environment, the gap between them remains intact and does not shrink. Consequently, the discriminator effectively emulates the sparse reward signal, providing dense reward signals to the RL agent. Intuitively, a state that is closer to the success states in terms of task progress (rather than Euclidean distance) receives a higher reward, as it is more likely to occur in success trajectories. Fig. 2.2(a) and (b) illustrate the distinction between our approach and traditional AIL methods.

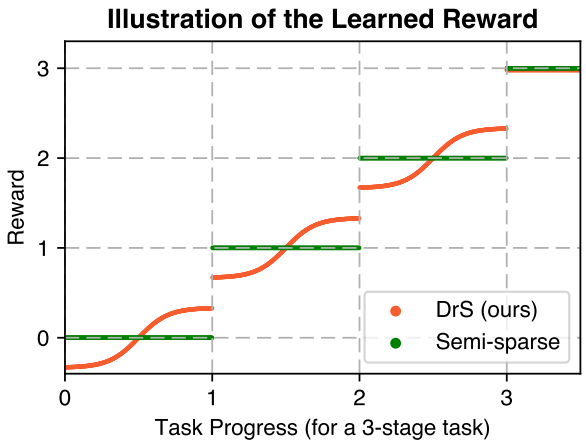

Figure 2.3. An illustration of our learned reward, which fills the gaps in semi-sparse rewards, resulting in a smooth reward curve.

To ensure that the training data consistently includes both success and failure trajectories, we use replay buffers to store historical experiences, and train the discriminator in an off-policy manner. While the original GAIL is on-policy, recent AIL methods [95, 145] have adopted off-policy training for better sample efficiency. Note that although our approach shares similarities with AIL methods, it is not adversarial in nature. In particular, our policy does not aim to deceive

the discriminator, and the discriminator does not seek to penalize the agent's trajectories.

2.4.2 Reward Learning on Multi-Stage Tasks

In multi-stage tasks, it is desirable for the reward of a state in stage $k+1$ to be strictly higher than that of stage k to incentivize the agent to progress towards later stages. The semi-sparse reward (Eq. 2.2) aligns with this intuition, but it is still a bit too sparse. If each stage of the task is viewed as an individual task, the semi-sparse reward acts as a sparse reward for each stage. In the case of a one-stage task, a discriminator can be employed to provide a dense reward. Similarly, for multi-stage tasks, a separate discriminator can be trained for each stage to serve as a dense reward for that particular stage. By training stage-specific discriminators, we can effectively address the sparse reward issue and guide the agent's progress through the different stages of the task. Fig. 2.3 gives an intuitive illustration of our learned reward, which fills the gaps in semi-sparse rewards, resulting in a smooth reward curve.

To train the discriminators for different stages, we need to establish the positive and negative data for each discriminator. In one-stage tasks, positive data comprises success trajectories and negative data encompasses failure trajectories. In multi-stage tasks, we adopt a similar approach with a slight modification. Specifically, we assign a stage index to each trajectory, which is determined as the highest stage index among all states within the trajectory:

$$\text{StageIndex}(\tau : (s_0, s_1, ...)) = \max_i \ \text{StageIndex}(s_i), \tag{2.3}$$

where τ is a trajectory and s_i are the states in τ. For the discriminator associated with stage k, positive data consists of trajectories that progress beyond stage k (StageIndex $> k$), and negative data consists of trajectories that reach up to stage k (StageIndex $\leq k$).

Once the positive and negative data for each discriminator have been established, the next step is to combine these discriminators to create a reward function. While the semi-sparse reward (Eq. 2.2) lacks incentives for the agent at stage k until it reaches stage $k+1$, we can fill in the gaps in the semi-sparse reward by the stage-specific discriminators. We define our learned

reward function for a multi-stage task as follows:

$$R(s') = k + \alpha \cdot \tanh(\text{Discriminator}_k(s')) \tag{2.4}$$

where k is the stage index of s' and α is a hyperparameter. Basically, the formula incorporates a dense reward term into the semi-sparse reward. The tanh function is used to bound the output of the discriminators. As the range of the tanh function is (-1, 1), any $\alpha < \frac{1}{2}$ ensures that the reward of a state in stage $k+1$ is always higher than that of stage k. In practice, we use $\alpha = \frac{1}{3}$ and it works well.

2.4.3 Implementation

From the implementation perspective, our approach is similar to GAIL, but with a different training process for discriminators. While the original GAIL is combined with TRPO [178], [145] found that using state-of-the-art off-policy RL algorithms (like SAC [65] or TD3 [54]) can greatly improve the sample efficiency of GAIL. Therefore, we also combine our approach with SAC.

In addition to the regular replay buffer used in SAC, our approach maintains N different stage buffers to store trajectories corresponding to different stages(defined by Eq. 2.3). Each trajectory is assigned to only one stage buffer based on its stage index. During the training of the discriminators, we sample data from the union of multiple buffers. In practice, we early stop the discriminator training of k once its success rate is sufficiently high, as we find it reduces the computational cost and makes the learned reward more robust. Note that our approach uses the next state s' as the input to the reward, which aligns with common practices in human reward engineering [64, 236]. However, our approach is also compatible with alternative forms of input, such as (s,a) or (s,a,s').

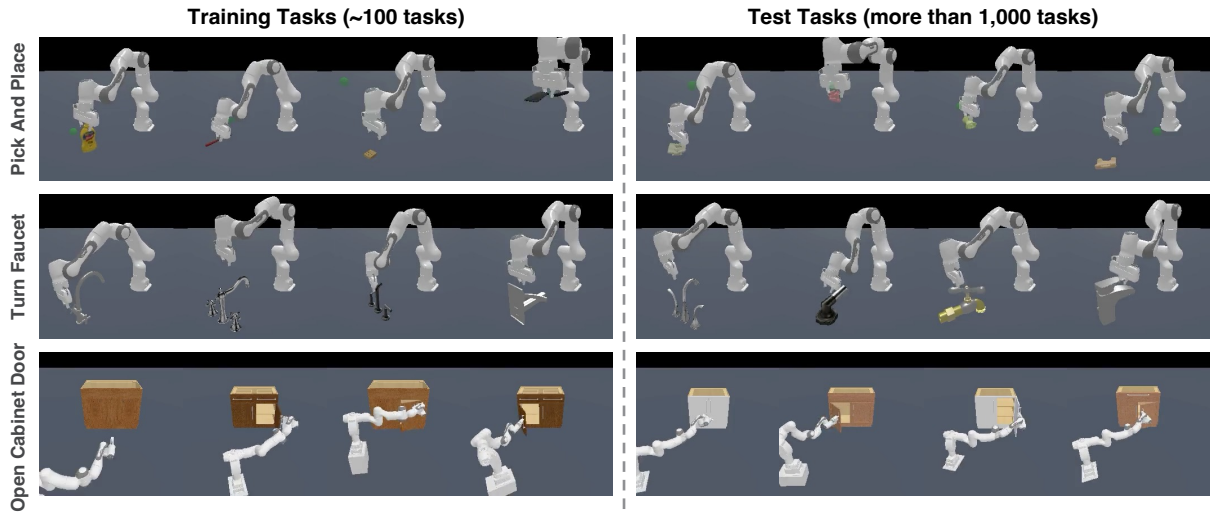

Figure 2.4. We evaluated our approach DrS on more than 1,000 task variants from three task families in ManiSkill [132, 64]. Each task variant is associated with a different object. All tasks require low-level physical control. The objects in training and test tasks are non-overlapped.

2.5 Experiments

2.5.1 Setup and Task Descriptions

We evaluated our approach on three challenging physical manipulation task families from the ManiSkill [132, 64]: Pick-and-Place, Turn Faucet, and Open Cabinet Door. Each task family includes a set of different objects to be manipulated. To assess the reusability of the learned rewards, we divided the objects within each task family into non-overlapping training and test sets, as depicted in Fig. 2.4. During the reward learning phase, we learned the rewards by training an agent for each task family to manipulate all training objects. In the subsequent reward reuse phase, the learned reward rewards are reused to train an agent to manipulate all test objects for each task family. And we compare with other baseline rewards in this **reward reuse** phase. It is important to note that our learned rewards are agnostic to the specific RL algorithm employed. However, we utilized the Soft Actor-Critic (SAC) algorithm to evaluate the quality of the different rewards.

To assess the *reusability* of the learned rewards, it is crucial to have a diverse set of tasks that exhibit similar structures and goals but possess variations in other aspects. However, most

existing benchmarks lack an adequate number of task variations within the same task family. As a result, we primarily conducted our evaluation on the ManiSkill benchmark, which offers a range of object variations within each task family. This allowed us to thoroughly evaluate our learned rewards in a realistic and comprehensive manner.

Pick-and-Place: A robot arm is tasked with picking up an object and relocating it to a random goal position in mid-air. The task is completed if the object is in close proximity to the goal position, and both the robot arm and the object remain stationary. The stage indicators include: (a) the gripper grasps the object, (b) the object is close the goal position, and (c) both the robot and the object are stationary. We learn rewards on 74 YCB objects and reuse rewards on 1,600 EGAD objects.

Turn Faucet: A robot arm is tasked to turn on a faucet by rotating its handle. The task is completed if the handle reaches a target angle. The stage indicators include: (a) the target handle starts moving, (b) the handle reaches a target angle. We learn rewards on 10 faucets and reuse rewards on 50 faucets.

Open Cabinet Door: A single-arm mobile robot is required to open a designated target door on a cabinet. The task is completed if the target door is opened to a sufficient degree and remains stationary. The stage indicators include: (a) the robot grasps the door handle, (b) the door is open enough, and (c) the door is stationary. We learn rewards on 4 cabinet doors and reuse rewards on 6 cabinet doors. Note that we remove all single-door cabinets in this task family, as they can be solved by kicking the side of the door and this behavior can be readily learned by sparse rewards.

We employed low-level physical control for all task families.

2.5.2 Baselines

Human-Engineered The original human-written dense rewards in the benchmark, which require a significant amount of domain knowledge, thus can be considered as an *upper bound* of performance.

Semi-Sparse The rewards are constructed based on the stage indicators, as discussed in Eq. 2.2. The agent receives a reward of k when it is in stage k. This baseline extends the binary sparse reward.

VICE-RAQ [186] An improved version of VICE [51]. It learns a classifier, where the positive samples are successful states annotated by querying humans, and the negative samples are all other states collected by the agent. Since our experiments do not involve human feedback, we let VICE-RAQ query the oracle success condition infinitely for a fair comparison.

ORIL [239] A representative offline reward learning method, where the agent does not interact with the environments but purely learns from the demonstrations. It learns a classifier (reward) to distinguish between the states from success trajectories and random trajectories.

2.5.3 Comparison with Baseline Rewards

We trained RL agents using various rewards and assessed the reward quality based on both the sample efficiency and final performance of the agents. The experimental results, depicted in Fig. 2.5, demonstrate that our learned reward surpasses semi-sparse rewards and all other reward learning methods across all three task families. This outcome suggests that our approach successfully acquires high-quality rewards that significantly enhance RL training. Remarkably, our learned rewards even achieve performance comparable to human-engineered rewards in Pick-and-Place and Turn Faucet.

Semi-sparse rewards yielded limited success within the allocated training budget, suggesting that RL agents face exploration challenges when confronted with sparse reward signals. VICE-RAQ failed in all tasks. Notably, it actually failed during the reward learning phase on the training tasks, rendering the learned rewards inadequate for supporting RL training on the test tasks. This failure aligns with observations made by [213]. We hypothesize that by only classifying the success states from other states, it cannot provide sufficient guidance during the early stages of training, where most states are distant from the success states and receive low rewards. Unsurprisingly, ORIL does not get any success on all tasks either. Without interacting

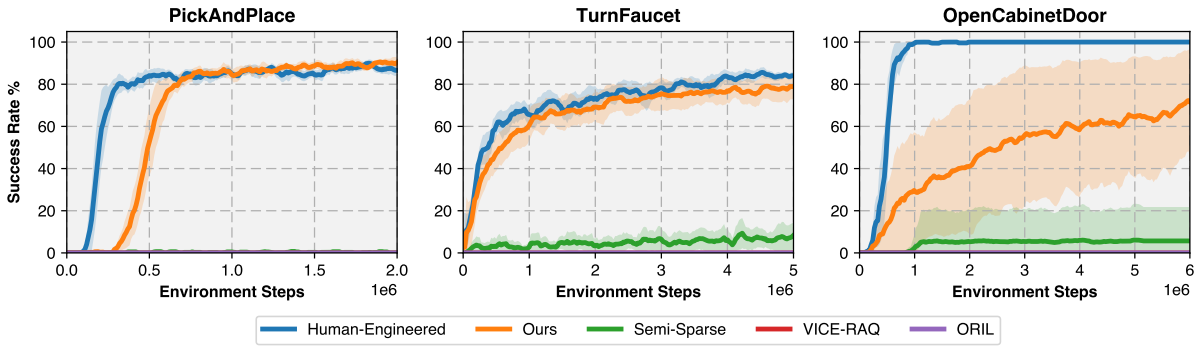

Figure 2.5. Evaluation results of **reusing learned rewards**. All curves use SAC to train but with different rewards. VICE-RAQ and ORIL had no success. 5 random seeds, the shaded region represents the standard deviation.

with the environments to gather more data, the learned reward functions easily tend to overfit the provided dataset. When using such rewards in RL, the flaws in the learned rewards are easily exploited by the RL agents.

2.5.4 Ablation Study

We examined various design choices within our approach on the Pick-and-Place task family.

Robustness to Stage Configurations

Though many tasks present a natural structure of stages, there are still different ways to divide a task into stages. To assess the robustness of our approach in handling different task structures, we experiment with different numbers of stages and different ways to define stage indicators.

Number of Stages

The Pick-and-Place task family originally consisted of three stages: (a) approach the object, (b) move the object to the goal, and (c) make everything stationary. We explored two ways of reducing the number of stages to two, namely merging stages (a) and (b) or merging stages (b) and (c), as well as the 1-stage case. Our results, presented in Fig. 2.6, indicate that

the learned rewards with 2 stages can still effectively train RL agents in test tasks, albeit with lower sample efficiency than those with 3 stages. Specifically, the reward that preserves stage (c) "make everything stationary" performs slightly better than the reward that preserves stage (a) "approach the object". This suggests that it may be more challenging for a robot to learn to stop abruptly without a dedicated stage. However, when reducing the number of stages to 1, the learned reward failed to train RL agents in test tasks, demonstrating the benefit of using more stages in our approach.

Definition of Stages

The stage indicator "object is placed" is initially defined as if the distance between the object and the goal is less than 2.5 cm. We create two variants of it, where the distance thresholds are 5cm and 10cm, respectively. The results, as depicted in Fig. 2.7, demonstrate that changing the distance threshold within a reasonable range does not significantly affect the efficiency of RL training. Note that the task success condition is unchanged, and our rewards consistently encourage the agents to reach the success state as it yields the highest reward according to Eq. 2.4. The stage definitions solely affect the efficiency of RL training during the reward reuse phase.

Overall, the above results highlight the robustness of our approach to different stage configurations, indicating that it is not heavily reliant on intricate stage designs. This robustness contributes to a significant reduction in the burden of human reward engineering.

Fine-tuning Policy

In our previous experiments, we assessed the quality of the learned reward by reusing it in training RL agents from scratch since it is the most common and natural way to use a reward. However, our approach also produces a policy as a byproduct in the reward learning phase. This policy can also be fine-tuned using various rewards in new tasks, providing an alternative to training RL agents from scratch. We compare the fine-tuning of the byproduct policy using human-engineered rewards, semi-sparse rewards, and our learned rewards.

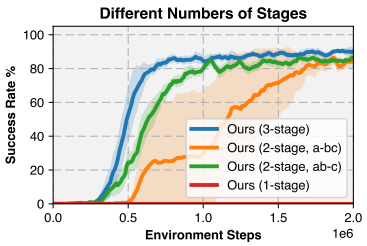

Figure 2.6. Ablation study on the number of stages, see here.

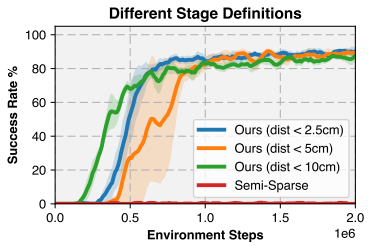

Figure 2.7. Ablation study on the stage definitions, see here for details.

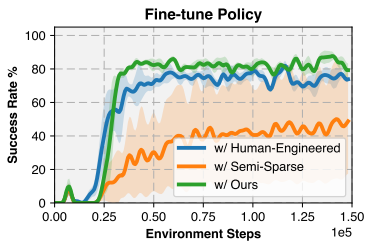

Figure 2.8. Fine-tune the policy from reward learning, see here.

As shown in Fig. 2.8, all policies improve rapidly at the beginning due to the good initialization of the policies. However, fine-tuning with our learned reward yields the best performance (even slightly better than the human-engineered reward), indicating the advantages of utilizing our learned dense reward even with a good initialization. Furthermore, the significant variance observed when fine-tuning the policy with semi-sparse rewards highlights the limitations of sparse reward signals in effectively training RL agents, even with a very good initialization.

2.6 Conclusion and Limitations

To make RL a more widely applicable tool, we have developed a data-driven approach for learning dense reward functions that can be reused in new tasks from sparse rewards. We have evaluated the effectiveness of our approach on robotic manipulation tasks, which have high-dimensional action spaces and require dense rewards. Our results indicate that the learned dense rewards are effective in transferring across tasks with significant variation in object geometry. By simplifying the reward design process, our approach paves the way for scaling up RL in diverse scenarios.

We would like to discuss two main limitations when using the multi-stage version of our approach.

Firstly, though our experiments show the substantial benefits of knowing the multi-stage structure of tasks (at training time, not needed at policy deployment time), we did not specifically investigate how this knowledge can be acquired. Much future work on be done here, by leveraging large language models such as ChatGPT [143] (by our testing, they suggest

stages highly aligned to the ones we adopt by intuition for all tasks in this work) or employing information-theoretic approaches.

Secondly, the reliance on stage indicators adds a level of inconvenience when directly training RL agents in the real world. While it is infrequent to directly train RL agents in the real world due to cost and safety issues, when necessary, stage information can still be obtained using existing techniques, similar to [87, 88]. For example, the "object is grasped" indicator can be acquired by tactile sensors [115, 124], and the "object is placed" indicator can be obtained by forward kinematics, visual detection/tracking techniques [87, 88], or even large vision-language models [39].

2.7 Acknowledgment

This chapter, in full, is taken from "DrS: Learning Reusable Dense Rewards for Multi-Stage Tasks" in International Conference on Learning Representations (ICLR) 2024 by Tongzhou Mu, Minghua Liu, Hao Su. The dissertation author was a primary investigator and author of this paper.

Chapter 3

Abstract-to-Executable Trajectory Translation for One-Shot Task Generalization

3.1 Introduction

Training long-horizon robotic policies in complex physical environments is important for robot learning. However, directly learning a policy that can generalize to unseen tasks is challenging for Reinforcement Learning (RL) based approaches [226, 176, 183, 133]. The state/action spaces are usually high-dimensional, requiring many samples to learn policies for various tasks. One promising idea is to decouple plan generation and plan execution. In classical robotics, a high-level planner generates an abstract trajectory using symbolic planning with simpler state/action space than the original problem while a low-level agent executes the plan in an entirely physical environment [85, 58]. In our work, we promote the philosophy of abstract-to-executable via a learning-based approach through RL. By providing robots with an abstract trajectory, robots can aim for *one-shot task generalization*. Instead of memorizing all the high-dimensional policies for different tasks, the robot can leverage the power of planning in the low-dimensional abstract space and focus on learning low-level executors.

The two-level framework works well for classical robotics tasks like motion control for robot arms, where a motion planner generate a kinematics motion plan at a high level and a PID controller execute the plan step by step. However, such a decomposition and abstraction is not always trivial for more complex tasks. In general domains, it either requires expert knowledge

(e.g., PDDL [58, 57]) to design this abstraction manually or enormous samples to distill suitable abstractions automatically (e.g., HRL [12, 210]). We refer [2] for an in-depth investigation into this topic.

On the other side, designing imperfect high-level agents whose state space does not precisely align with the low-level executor could be much easier and more flexible. High-level agents can be planners with abstract models and simplified dynamics in the simulator (by discarding some physical features, e.g., enabling a "magic" gripper [176, 201]) or utilizing an existing "expert" agent such as humans or pre-trained agents on different manipulators. Though imperfect, their trajectories still contain meaningful information to guide the low-level execution of novel tasks. For example, different robots may share a similar procedure of reaching, grasping, and moving when manipulating a rigid box with different grasping poses. As a trade-off, executing their trajectories by the low-level executors becomes non-trivial. As will be shown by an example soon, there may not be a frame-to-frame correspondence between the abstract and the executable trajectories due to the mismatch. Sometimes the low-level agent needs to discover novel solutions by slightly deviating from the plan in order to follow the rest of the plan. Furthermore, the dynamics mismatch may require low-level agents to pay attention to the entire abstract trajectory and not just a part of it.

To benefit from abstract trajectories without perfect alignment between high and low-level states, we propose **TR**ajectory **TR**anslation (abbreviated as **TR**^{2}), a learning-based framework that can translate abstract trajectories into executable trajectories on unseen tasks at test time. The key feature of **TR**^{2} is that *we do not require frame-to-frame alignment* between the abstract and the executable trajectories. Instead, we utilize a powerful sequence-to-sequence translation model inspired by machine translation [193, 13] to translate the abstract trajectories to executable actions even when there is a significant domain gap. This process is naturally reminiscent of language translation, which is well solved by seq-to-seq models.

We illustrate the idea in a simple Box Pusher task as shown in Fig. 3.1. The black agent needs to push the green target box to the blue goal position. We design the high-level agent as a

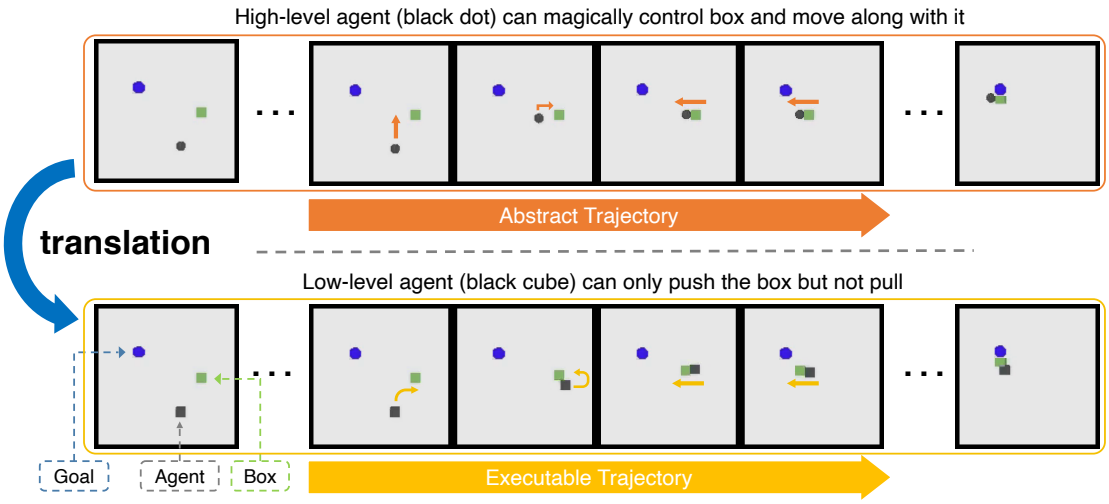

Figure 3.1. Task in Box Pusher: move the green target box to the blue goal position. The arrows in map show how the agents move.

point mass which can magically attract the green box to move along with it. For the high-level agent, it is easy to generate an abstract trajectory by either motion planning or heuristic methods. As **TR**^{2} does not have strict constraints over the high-level agent, we can train **TR**^{2} to translate the abstract trajectory, which includes the waypoints to the target, into a physically feasible trajectory. Our **TR**^{2} framework learns to translate the magical abstract trajectory to a strategy to move around the box and push the box to the correct direction, which closes the domain gap between the high- and low-level agents.

Our contributions are: **(1)** We provide a practical solution to learn policies for long-horizon complex robotic tasks in three steps: build a paired abstract environment by using point mass representations with magical grasping as the high-level agent, generate abstract trajectories, and solve the original task with abstract-to-executable trajectory translation.

(2) The seq-to-seq models, specifically the transformer-based auto-regressive model [205, 27, 147], free us from the restriction of strict alignment between abstract and executable trajectories making abstract trajectory generation easier and helps bridge the domain gap.

(3) The combination of the abstract trajectory and transformer enables **TR**^{2} to solve unseen long-horizon tasks. By evaluating our method on a navigation-based task and three

manipulation tasks, we find that our agent achieves strong one-shot generalization to new tasks, while being robust to intentional interventions or mistakes via re-planning.

Our method is evaluated on various tasks and environments with different embodiments in state-based settings. In all experiments, the method shows great improvements over baselines. We also perform real-world experiments on the Block Stacking task to verify the capability to handle noise on a real robot system.

3.2 Related Works

One-Shot Imitation Learning. Recent studies [40, 46, 149, 225, 234, 225, 118, 190] have shown that it is feasible to teach a robot new skills using only a single demonstration, akin to how humans acquire a wide variety of abilities with a single demonstration. To achieve one-shot generalization, these works usually assume a dataset of expert demonstrations, which is used to train a behavior cloning model [40, 168, 201, 80] that accelerates learning in future novel tasks. In terms of architecture, [220] is similar to us but still requires a dataset of low-level demos and is experimented in the offline setting whereas we utilize a dataset of simpler abstract trajectories and train online in complex environments.

Cross-Morphology Imitation Learning. When there is a morphology difference between expert and imitator, a manually-designed retargeting mapping is usually used to convert the state and action for both locomotion [151, 4] and manipulation [191, 162, 10]. However, the mapping function is task-specific, which limits the application of these approaches to a small set of tasks. To overcome this limitation, action-free imitation is explored by learning a dynamics model [201, 167, 116, 42] to infer the missing action or a reward function [11, 228, 181] to convert IL to a standard RL paradigm. However, these methods need extensive interaction data to learn the dynamics model or update policy with learned rewards. Instead of learning the reward function from cross-morphology teacher trajectories, we propose a generic trajectory following reward based on the given abstract trajectory that can provide dense supervision for the agent.

DeepMimic [150] is similar to ours, but it differs in that we do not allow training at inference time with new trajectories . SILO [102] is another similar approach that trains an agent to follow the demonstrator but is limited by fixed window sizes/horizon parameters that inhibit the test generalizability of its approach.

Demo Augmented Reinforcement Learning. Motion primitives, especially dynamic motion primitives, have long been used by robotics researchers to combine the human demonstration with RL [92, 198, 113, 185]. Recently, [153, 152] extended this framework to learn task-agnostic skills from demonstrations. Another line of work [72, 169] directly use demonstrations as interaction data. For example, Demo Augmented Policy Gradient [169, 167] performs behavior cloning and policy gradient interchangeably with a decayed weight for imitation in on-policy training while other works [207, 71] append the demonstrations into the replay buffer for off-policy RL. However, their works typically utilize low-level demonstrations while we utilize abstract trajectories that are much easier to generate especially for novel tasks.

Hierarchical Reinforcement Learning. HRL presents an intuitive approach to tackling complex multi-layered problems by using an hierarchical structure in task solving. One common approach to HRL is to simultaneously learn a high-level policy that generates sub-goals and learn a low-level policy to follow these sub-goals [210, 108, 135]. However these end-to-end learning methods in HRL are difficult to train for more complex, long-horizon tasks without strong reward signals. As a result some approaches seek to integrate explicit high-level planning mechanisms to guide the low-level policy trained via RL [60, 78]. Our approach has similarities to HRL's hierarchical task decomposition but does not learn a high-level policy, making it easier to train. Furthermore, the use of transformers for translation further enable deeper long-horizon reasoning necessary for some complex tasks as investigated in our experiments.

3.3 Method

3.3.1 Overview and Preliminaries

Similar to one-shot imitation learning [40], we are tackling the problem of one-shot task generalization. In one-shot imitation learning, an agent must solve an unseen task given a demonstration (e.g., human demo, low-level demo) without additional training. However, even a single demonstration can be challenging to produce, especially for complex long-horizon robotics tasks. Different from one-shot imitation learning, we replace the demonstration with an abstract trajectory. The abstract trajectory is a sequence of high-level states corresponding to a high-level agent that instructs the low-level agent on how to complete the task at a high level. In the high-level space, we strip low-level dynamics and equip the high-level agent with magical grasping, allowing it to manipulate objects easily. Moreover, the high-level space converts all relevant objects to point masses. The simplification makes abstract trajectories easier and more feasible to generate for long-horizon unseen tasks compared to human demos or low-level demos.

Given a novel task, our method seeks to solve it with the following three steps: (i) construct a paired abstract environment with point mass representation that can be solved with simple heuristics or planning algorithms and generate abstract trajectories (Sec. 3.3.2); (ii) translate the high-level abstract trajectory to a low-level executable trajectory with a trained trajectory translator in a closed loop manner (Sec. 3.3.3, 3.3.4); (iii) solve the given task and potentially other unseen tasks with the trajectory translator (Sec. 3.3.5). The three steps above enable our approach to tackle unseen long-horizon tasks that are out-of-distribution as well. Moreover, we can utilize the re-planning feature (regenerating the abstract trajectory during an episode) to increase the success rate at test time to handle unforeseen mistakes or interventions.

Next, we introduce definitions and symbols. We consider an environment as a Markov Decision Process (MDP) $(\mathscr{S}^L, \mathscr{A}, \mathscr{P}, \mathscr{R})$, where $\mathscr{A}$ is the action space, $\mathscr{P}$ is the transition function of the environment, and $\mathscr{R}$ is the reward function. Different from the regular MDP, we consider two state spaces, the low-level state space $\mathscr{S}^L$ and a high-level state space $\mathscr{S}^H$. We

assume there exists a map $f : \mathscr{S}^L \to \mathscr{S}^H$ and a dissimilarity function $d : \mathscr{S}^L \times \mathscr{S}^H \to \mathbb{R}$ such that $d(s^L, s^H) = 0$ for $s^H = f(s^L)$, where $s^H \in \mathscr{S}^H$ and $s^L \in \mathscr{S}^L$. The high-level agent generates an abstract trajectory $\tau^H = (s_1^H, s_2^H ..., s_T^H)$ from an initial state $s_1^H = f(s_1^L)$. Note that actions are not included in τ^H. Lastly, the low-level agent receives observations s_t^L in the low-level state space $\mathscr{S}^L$, and takes actions a_t in the action space $\mathscr{A}$, and the dynamics $\mathscr{P}$ returns the next observation $s_{t+1}^L = P(s_t^L, a_t)$.

3.3.2 Abstracting Environments and Generating Abstract Trajectories

The first step to solving a challenging task is to build a paired task that abstracts the low-level physical details away. The paired task should be much simpler than the original task so that it can be solved easily. We leverage two general ways to build the abstract environment: (i) simplify geometry [123], e.g., representing the agent and objects with point masses; (ii) abstract contact dynamics [189, 94], e.g., the original environment requires detailed physical manipulation to grasp objects while the agent in the abstract environment can magically grasp the objects. Thus, leveraging the two methods above, a concrete solution to construct abstract environments for many difficult tasks can be done.

Concretely for all our abstract environments we remove all contact dynamics, enable magical grasp, and represent all relevant objects as point masses. The point mass representation further makes the mapping function f simple to define. As a result, making the abstract environment is scalable as we use the same simple process for all environments. This then enables simple generation of abstract trajectories with heuristics with a point mass high-level agent. To generate abstract trajectories, for manipulation tasks, we make the high-level agent approach the object, then magically grasp the object, and then finally move the object to the target position. If there are no objects to be manipulated in the task, then it is a simple navigation sequence.

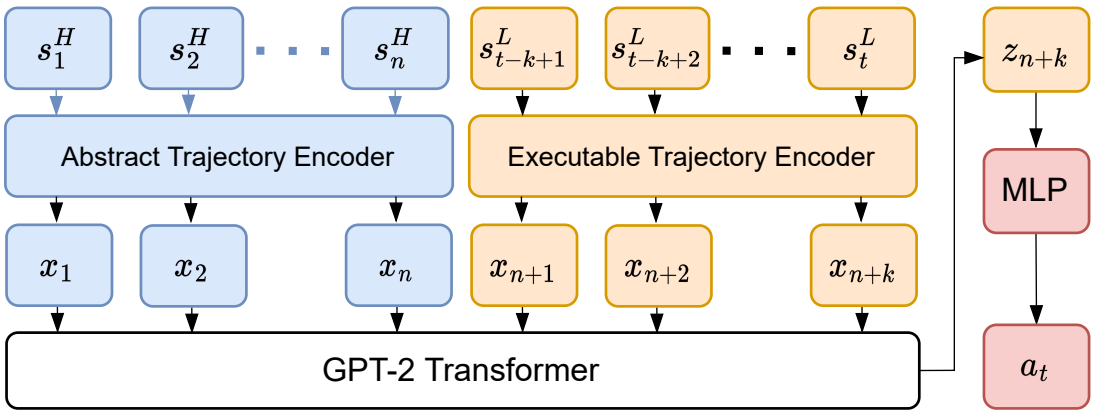

Figure 3.2. Illustration of the abstract-to-executable trajectory translation architecture. High-level states $s_{H,i}$ are fed through one encoder and the low-level states $s_{L,i}$ are fed through a separate encoder to create tokens. The tokens form a sequence that is given to the transformer model, and the final output embedding z_{n+k-1} is passed through an MLP to produce actions.

3.3.3 Abstract-to-Executable Trajectory Translator

To translate abstract trajectories into executable low-level actions, we aim to learn a low-level policy $\pi_\theta^L(a_t^L|s_t^L, s_{t-1}^L ..., s_{t-k+1}^L, \tau^H)$ that utilizes the current and past k low-level states and an abstract trajectory τ^H generated by a high-level agent. An overview of our translation model and the flow of inputs and outputs is shown in Fig. 3.2.

The environment at time step t returns to the low-level agent the current and past low-level states $s_t^L, s_{t-1}^L ..., s_{t-k+1}^L$ for context size k. The high-level agent further provides an abstract trajectory $\tau^H = (s_1^H, s_2^H, ..., s_n^H)$ composed of high-level states s_i^H which can also be viewed as a prompt. However, due to the high-level nature of the abstract trajectory and lack of a frame-to-frame correspondence, learning to follow the abstract trajectory requires a deeper understanding about the abstract trajectory.

Thus, we adopt the transformer architecture, specifically GPT-2 [166, 21, 165], and we format the input sequence by directly appending the current low-level state and past low-level states to the abstract trajectory. With the transformer's attention mechanism [205], when processing the current low-level state s_t^L, the model can attend to past low-level states as well as the entire abstract trajectory to make a decision. By allowing attention over the entire abstract

trajectory, our model is capable of modelling long-horizon dependencies better and suffers less from information bottlenecks. For example, in Box Pusher (Fig. 3.1) the low-level agent must look far ahead into the future in the abstract trajectory to determine which direction the high-level agent (black dot) moves the green target box. By understanding where the target box moves, the low-level agent can execute the appropriate actions to position itself in a way to move the target box the same way and follow the abstract trajectory.

Note that, while the backbone of the model discussed here is the GPT-2 transformer, it can easily be replaced by any other seq-to-seq model like an LSTM [74].

3.3.4 Training with Trajectory Following Reward

$$R^{Traj} = \begin{cases} 0 & \text{if } j_t < n \text{ and } j'_t \leq j_{t-1} \text{ (make no progress)} \\ (1+\beta \cdot j'_t) \cdot r_{dist}(s^L_t, s^H_{j'_t}) & \text{if } j_t < n \text{ and } j'_t > j_{t-1} \text{ (make progress)} \\ r_{dist}(s^L_t, s^H_n) & \text{if } j_t = n \text{ (has matched all high-level states)} \end{cases} \tag{3.1}$$

Conventionally, seq-to-seq models are trained on large parallel corpora for translation tasks like English to German [117] in an auto-regressive / open loop manner. However, we desire to train a policy network that solves robotic environments well. To this end, we adapt the seq-to-seq model to a closed-loop setting where the model receives environment feedback at every step as opposed to an open-loop setting, reducing error accumulation that often plagues pure offline methods. Thus, to train the translation model described in Sec. 3.3.3, we use online RL to maximize a trajectory following reward (3.1), and specifically, we use the PPO algorithm [179]. Note that our framework is not limited to any particular algorithm, and our framework can also work in offline settings if an expert low-level dataset is available.

The core idea of the trajectory following reward is to *encourage the low-level agent to match as many high-level states in the abstract trajectory as possible*. We say a low-level state s^L matches a high-level state s^H when s^H has the shortest distance to s^L, and this distance is lower than a threshold ε. During an episode, we track the farthest high-level state the low-level agent

has matched and use j_t to denote the index of the farthest high-level state that has been matched at timestep t.

Concretely $j_t = \max_{1 \leq k \leq t} j'_k$, where

$$j'_k = \text{argmin}_{1 \leq i \leq n} \{ d(s_k^L, s_i^H) | d(s_k^L, s_i^H) < \varepsilon \}$$

is the index of the high-level state which is matched by the low-level state s_k^L, n is the length of the abstract trajectory, and $d(s_t^L, s_{j'}^H) = ||f(s_t^L) - s_{j'}^H||$. We define our trajectory following reward in equation 3.1.

In equation 3.1, $r_{dist}(s_t^L, s_{j'}^H) = 1 - \tanh(w \cdot d(s_t^L, s_{j'}^H))$ is a common distance-based reward function which maps a distance to a bounded reward, and the weight term $(1 + \beta \cdot j'_t)$ is used to emphasize more on the later, more difficult to reach, high-level states. w and β are scaling hyperparameters but are kept the same for all environments and experiments.

In practice, we combine the trajectory following reward with the original task reward in order to improve the training efficiency of our method.

Note that the original task rewards are simplistic and not always advanced enough such that a goal-conditioned policy can solve the task.

Furthermore, a limitation of the reward function is that it cannot handle abstract trajectories when a high-level state is repeated. This presents a problem in periodic tasks such as pick and placing a block between two locations repeatedly.

3.3.5 Test with Trajectory Translation and Re-Planning

At test time starting with low-level state s_1^L, we generate an abstract trajectory τ^H from the mapped initial high-level state $s^H = f(s_1^L)$. At timestep t, we execute our low-level agent π_θ^L in a closed-loop manner by taking action $a_t = \text{argmax}_{a \in \mathscr{A}} \pi_\theta^L(a | s_t^L, s_{t-1}^L ..., s_{t-k+1}^L, \tau^H)$. In addition, we are able to re-generate the abstract trajectory in the middle of completing the task to further boost the test performance and handle unforeseen situations such as external interventions

or mistakes. We refer to this strategy as *re-planning* and investigate it in Sec. 3.4.6. Note that re-planning is not adopted in most one-shot imitation learning methods because low-level or human demonstrations are challenging and time-consuming to generate for new tasks and are impractical for long-horizon tasks. In contrast, with a high-level agent acting in a simple high-level state space, we can alleviate these problems and quickly generate abstract trajectories to follow at test time.

In practice, we run re-planning in one of the two scenarios: 1) If the agent matches the final high-level state in τ^H, we will re-plan to begin solving the next part of a potentially long horizon task as done for Block Stacking and Open Drawer test settings. 2) If after maximum timesteps allowed, the agent has yet to match the final high-level state, then we will re-plan as the agent likely had some errors. Re-planning enables the low-level policy to solve tasks even when there is some intervention or mistake, in addition to allowing it to solve longer-horizon tasks.

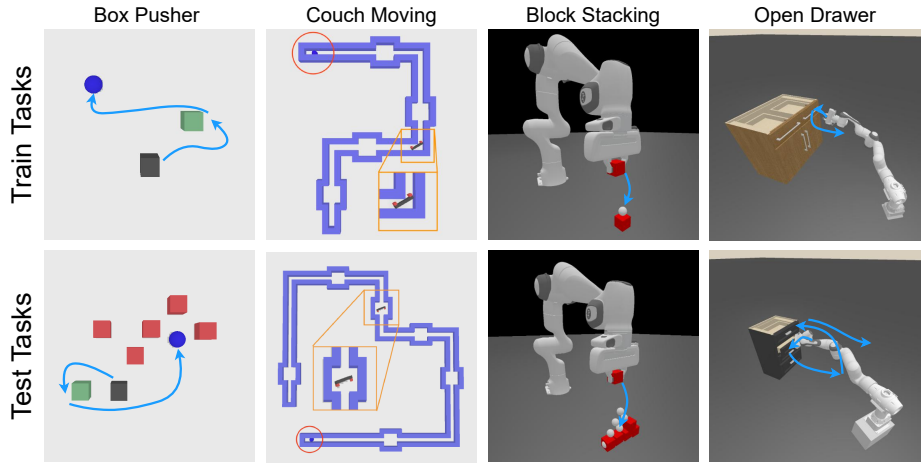

Figure 3.3. Training and Test Environments, with arrows indicating the direction of movement

3.4 Experiments

The effectiveness of **TR**^{2}-GPT2 (our **TR**^{2} method with a GPT2 backbone) originates from two key designs: abstract trajectory setup and the complementing transformer architecture.

These two designs contribute to strong performances, especially for long-horizon and unseen tasks.

To evaluate our approach, our experiments will answer the following questions:

- How does **TR**^{2}-GPT2 performs compared to other baselines? (Sec. 3.4.2)
- How does **TR**^{2}-GPT2 perform on long-horizon unseen tasks? (Sec. 3.4.3)
- How does the abstract trajectory setup impact learning? (Sec. 3.4.4)
- How does **TR**^{2}-GPT2 translate trajectories to bridge the domain gap via attention? (Sec. 3.4.5)
- How can re-planning improve performance in long-horizon tasks? (Sec. 3.4.6)

To answer these questions, we build four robotic tasks in the SAPIEN [217] simulator with realistic low-level dynamics shown in Fig. 3.3. These environment support flexible configurations that can generate task variants with different horizon lengths. The Box Pusher task is the simplest and can be used for fast concept verification. The Couch Moving task can test long-horizon dependency and abstract trajectory length generalization. The last two tasks, Block Stacking and Open Drawer, have full-physics robot embodiment with visual sensory input, which can evaluate the performance and generalizability of our method on more difficult high-dimensional robotics tasks. For each environment, we build a paired environment with point mass based simplifications and magical grasping to generate abstract trajectories via heuristic methods.

We compare our method, namely **TR**^{2}-GPT2, with three baselines on our four environments.

- **TR**^{2}-LSTM: Similar to our method, but replaces GPT-2 with an LSTM [74].
- **Goal-conditioned Policy (GC)**: Instead of using an abstract trajectory as guidance, it receives a task-specific goal and the original task reward, implemented as an MLP.

- **Subgoal-conditioned Policy (SGC)**: Similar to GC, but it receives a sub-goal, which is a high-level state n steps ahead of the current furthest matched high-level state, and is trained with the same reward as **TR**^{2}-GPT2. The correspondence between low- and high-level states are matched using the algorithm in Sec. 3.3.4.

Furthermore, we compare against a similar line of work, SILO [102], and evaluate our method on two of their environments: Obstacle Push and Pick and Place (from SILO).

Table 3.1. Mean success rate and standard error of training and test tasks, evaluated over 5 training seeds and 128 evaluation episodes each. **TR**^{2}-GPT2 outperforms other baselines, especially on test scenarios. Due to a difficult success metric, both **TR**^{2}-LSTM and GC could only partially solve block stacking and GC could only partly solve Open Drawer.

	Task	**TR**^{2}-GPT2	**TR**^{2}-LSTM	SGC	GC
1	Box Pusher (Train)	77.5±1.6	39.2±1.9	45.7±2.0	**82.0±1.5**
2	Box Pusher w/ Obstacles	**63.9±1.9**	22.0±1.6	19.8±1.6	48.3±2.0
3	Couch Moving Short 3 (Train)	**86.7±1.3**	59.5±1.9	21.3±1.6	7.0±1.0
4	Couch Moving Long 3	**81.3±1.5**	58.8±1.9	39.2±1.9	0.6±0.3
5	Couch Moving Long 4	**73.4±1.7**	44.4±2.0	20.9±1.6	0.5±0.3
6	Couch Moving Long 5	**58.3±1.9**	31.6±1.8	12.5±1.3	0±0
7	Pick and Place (Train)	**98.3±0.5**	0±0	57.0±2.0	0±0
8	Stack 4 Blocks	**92.5±1.0**	0±0	0±0	0±0
9	Stack 5 Blocks	**68.1±3.7**	0±0	0±0	0±0
10	Stack 6 Blocks	**33.8±3.7**	0±0	0±0	0±0
11	Build 3-2-1 Pyramid	**93.8±1.9**	0±0	0±0	0±0
12	Build 4-3-2-1 Pyramid	**80.0±3.2**	0±0	0±0	0±0
13	Open 1 Drawer (Train)	**90.0±2.4**	37.3±3.8	61.3±3.9	0±0
14	Open 1 Unseen Drawer	**87.0±2.7**	40.9±3.9	57.7±3.9	0±0
15	Open 2 Drawers	**18.1±3.0**	15.6±6.9	7.5±2.1	0±0

Table 3.2. Mean success rate and standard error from evaluating over 3 training seeds and 128 evaluation episodes each, compared with [102].

Task	**TR**^{2}-GPT2	SILO
Obstacle Push	95.3±1.1	70.0
Pick and Place	100±0	95.0

3.4.1 Environment Details

This section outlines all experimented environments, detailing the train and test tasks.

Box Pusher: The goal is to control a black box to push a green box towards a blue goal. The low-level agent is restricted to only pushing the green box whereas the high-level agent can magically grasp the green box and pull it along the way. In training there are no red obstacles but these are added at test time which block the low-level agent. The low-level agent will fail if it cannot follow the abstract trajectory precisely.

Couch Moving: The goal is to control a couch shaped agent to complete a maze composed of chambers and corners. The couch shape forces the low-level agent to rotate in chambers ahead of time in order to turn around corners, akin to the moving-couch problem. On the other hand the high-level agent is a point mass that can easily traverse the entire maze. The naming convention for these tasks is Couch Moving [Short/Long] n where Couch Moving Short 3 is the training task. Short/Long represents the lengths between chambers and corners (Long is about 1.5x longer than Short), and n is the number of corners the agent must turn. The low-level agent can only observe itself and a local patch around it, but does not have direct access to the full maze configuration. As a result, the test settings test whether the low-level agent can process abstract trajectories to determine when to rotate and generalize to longer abstract trajectory lengths for test mazes.

Block Stacking: The goal is to control a 8-DoF Panda robot arm (w/ gripper) to pick up a block, place it at a goal position, and then return to a resting position.

During training, the agent only needs to pick and place a single block up to a height of 3 blocks. At test time, the agent needs to deal with multiple blocks as well stack various configurations like a tower or pyramid of increasing heights. This setting tests whether the low-level agent can follow out-of-distribution abstract trajectories not seen during training, e.g. stacking higher, placing further away. It also evaluates the performance on much longer horizon tasks.

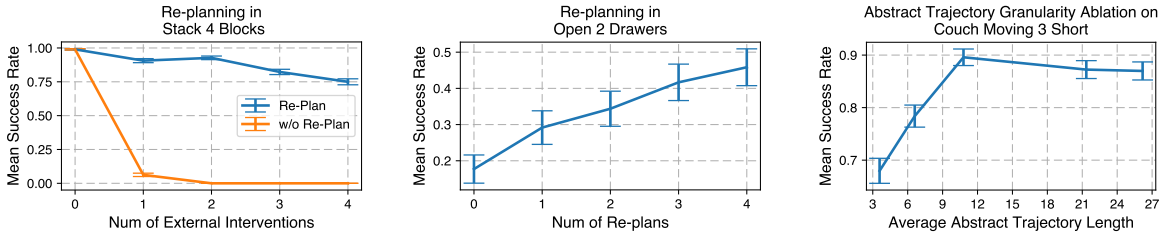

Figure 3.4. Evaluation of **TR**^{2}-GPT2 on **(left)** stacking 4 blocks with varying amounts of external interventions, only re-planning once per intervention; **(middle)** opening two drawers with variable amounts of re-planning; **(right)** Couch moving with varying granularity of abstract trajectories.

Open Drawer: The goal is to control a 13-DoF mobile robot (w/ gripper) to open a drawer on various cabinets. During training the task is to pull open a drawer on cabinets from a training set. At test time, the task is to pull open drawers on unseen cabinets and / or open additional drawers in an episode. The test setting tests whether the low-level agent can learn to follow the abstract trajectory and manipulate unseen drawer handles.

Obstacle Push and Pick and Place from SILO [102]: We recreate the two tasks from SILO (they did not release code) and compare our method against SILO on these tasks. The Obstacle Push environment consists of a high-level agent (or demo in SILO) magically pushing an object through an obstacle in the environment, with the task of the low-level agent to push the object around the obstacle to get to the same goal. The Pick and Place environment consists of a high-level agent magically picking and placing the block while going through one of two walls on the side. The low-level agent must do the same with physical manipulation and cannot go through the two walls. These tasks demonstrate examples of the high-level agent performing trajectories infeasible to the low-level agent and thus requiring the low-level agent to only follow what is feasible.

3.4.2 Results and Analysis

As shown in Table 3.1, our **TR**^{2}-GPT2 performs better than all other baselines, especially on test tasks that are unseen and long-horizon. The performance can be attributed to the abstract

trajectory setup of **TR**^{2} and the transformer architecture, which is further investigated in ablation studies.

The GC baseline cannot solve most tasks as the designed task rewards do not provide sufficient guidance. For the SGC baseline, even when conditioned with sub-goals it still has low success rates on test tasks, indicating that simple heuristics are not sufficient to select good subgoals.

The **TR**^{2}-LSTM baseline also performs worse than **TR**^{2}-GPT2. One interpretation is that modelling long-horizon dependencies with LSTMs is challenging due to the information bottleneck. This prompts us to investigate how the transformer's attention module intuitively leverages the long-term information in Sec. 3.4.5. Lastly, compared to SILO, which seeks to imitate a demo as closely as possible like us, we achieve better results shown in Table 3.2.

3.4.3 Performance on Long-Horizon Unseen Tasks

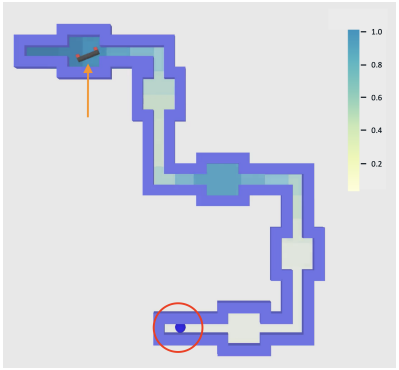

Figure 3.5. In this figure, agent attends to its current location as well as the second next chamber.

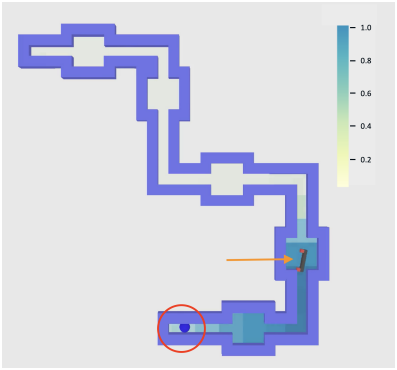

Figure 3.6. Agent attends to the current location, the next corner and chamber, but none of the locations it has passed already.

In Couch Moving, we test the model's ability to generalize to long-horizon unseen tasks with out-of-distribution abstract trajectories. Rows 3-6 in Table 3.1 demonstrate how the model can successfully solve much longer and varying mazes, showcasing the expressive power of **TR**^{2}-GPT2 compared to baselines. These experiments also show that our model can handle variable sequence lengths and horizons at test time. In comparison, due to fixed/manually tuned

horizon sizes, SILO and hierarchical RL methods such as Hierarchical Actor Critic [108] have difficulty handling the variance in abstract trajectory lengths.

Lastly, in Open Drawer we test the ability of **TR**^{2}-GPT2 to generalize to opening drawers of different geometries and at different locations. Row 14 of Table 3.1 shows that **TR**^{2}-GPT2 can generalize well beyond the other baselines.

3.4.4 How Does the Granularity of the Abstract Trajectory Impact Learning

We perform an ablation study where we vary the granularity of the abstract trajectory, with the most sparse settings resembling closer to that of single-goal conditioned policies and denser settings resembling that of imitation learning by imitating each frame. Results in Fig. 3.4 (right) show as the abstract trajectory becomes less granular (more sparse), the more difficult it is for agents to learn to follow the abstract trajectory and solve the task. In the Couch Moving task for example, an insufficient number of high-level states makes the problem ambiguous and more difficult to determine the correct orientation to rotate into, causing the agent to get stuck at corners frequently. Thus, the results show that low-level policies trained with sufficiently granular abstract trajectories can be successful.

3.4.5 Attention Analysis of TR2-GPT2 for Bridging the Domain Gap

In general, **TR**^{2}-GPT2 will learn whatever is necessary to bridge the domain gap between the high-level and low-level spaces and fill in gaps of information that are excluded from the high-level space. For example, in Couch Moving, the abstract trajectory does not include information about when and how to rotate the couch, only a coarse path to the goal location. Thus the low-level policy must learn to rotate the couch appropriately to go through the corners and follow the abstract trajectory. To visually understand how **TR**^{2}-GPT2 learns to bridge the domain gap, we investigate the attention of the transformer when solving the Couch Moving task.

We observe that after training, **TR**^{2}-GPT2 exhibits an understanding of an optimal

strategy to determine when to rotate or not. As shown in Fig. 3.5 and 3.6, whenever the agent is in a chamber that permits rotation, the agent attends to around its current position as well as a location in front of it. The attention on its current location tells the agent the orientation of the chamber (if it's up/down or left/right). The attention on a future corner/chamber, in combination with the information about the orientation of the chamber the agent is currently in, is indicative of the orientation of the upcoming corner. Attending to these locations enables the agent to successfully bridge the high- to low domain gap in Couch Moving. Moreover, the agent learns to pay attention mostly to locations up ahead and learns that the past parts are uninformative, despite being given the full abstract trajectory to process at each timestep. A video of the attention of the agent over the course of the episode can be found on our project page.

3.4.6 Effects of Re-Planning

One property of our approach is the feasibility of re-generating the abstract trajectory at test time, and we refer to this as *re-planning*. It enables us to introduce explicit error-corrective behavior via the high-level agent in long-horizon tasks. Our results in Fig. 3.4 (left and middle) show that re-planning enables handling unforeseen interventions in Block Stacking and mistakes in Open Drawer successfully.

In Block Stacking, we add external interventions where we randomly move blocks off the tower and allow the agent to re-plan just once per intervention. As the number of external interventions increases, the success rate does not lower as significantly compared to not re-planning since the re-generated abstract trajectories guide the low-level agent to pick up misplaced blocks.

In Open Drawer, the robot arm must open two drawers and often will close the one it opened by accident. With re-planning, the high-level agent provides a corrective abstract trajectory to guide the low-level agent to re-open the closed drawer and succeed.

3.5 Conclusion

We have introduced the Trajectory Translation ($\mathbf{TR}^2$) framework that seeks to train low-level policies by translating an abstract trajectory into executable actions. As a result we can easily decouple plan generation and plan execution, allowing the low-level agent to simply focus on low-level control to follow an abstract trajectory. This allows our method to generalize to unseen long-horizon unseen tasks. We further can utilize re-planning via the high-level agent to easily improve success rate to handle situations when mistakes or external intervention occurs. These results are exemplified by our simulation and real-world experiments. While the point mass representation works well for all benchmarked environments, it may have limitations when tackling environments such as soft-body environments and may need a different choice of representation. We believe there is interesting future work in investigating the design of abstract environments and how they may impact the training and results of $\mathbf{TR}^2$. Moreover, using offline demonstration datasets may also be interesting to investigate in order to improve the online sample efficiency and solve more complex tasks with state or visual observations.

3.6 Acknowledgment

This chapter, in full, is taken from "Abstract-to-Executable Trajectory Translation for One-Shot Task Generalization" in International Conference on Machine Learning (ICML) 2023 by Stone Tao, Xiaochen Li, Tongzhou Mu, Zhiao Huang, Yuzhe Qin, Hao Su. The dissertation author was an author of this paper.

Chapter 4

Policy Decorator: Model-Agnostic Online Refinement for Large Policy Model

4.1 Introduction

Encouraged by the recent success of large language and vision foundation models [21, 91], the field of robot learning has seen significant advances through **imitation learning** (particularly behavior cloning), where large models leverage extensive robotic demonstrations to develop effective policies [16, 19, 18, 5]. Despite these advancements, the performance of learned models is limited by the quantity, quality, and diversity of pre-collected demonstration data. This limitation often prevents models from handling all potential corner cases, as *demonstrations cannot cover every possible scenario (e.g., test-time objects can be entirely different from training ones)*. Unlike NLP and CV, scaling up demonstration collection in robotics, such as RT-1 [19] and Open X-Embodiment [33], requires extensive time and resources, involving years of data collection by numerous human teleoperators, making it costly and time-consuming. In contrast, cognitive research indicates that infants acquire skills through active interaction with their environment rather than merely observing others [177, 3, 35]. This raises a natural question: *Can we further improve an offline-trained large policy through online interactions with the environment?*

The most straightforward approach to improving an offline-trained imitation learning policy is to fine-tune it using reinforcement learning (RL) with a sparse reward [97, 223]. However,

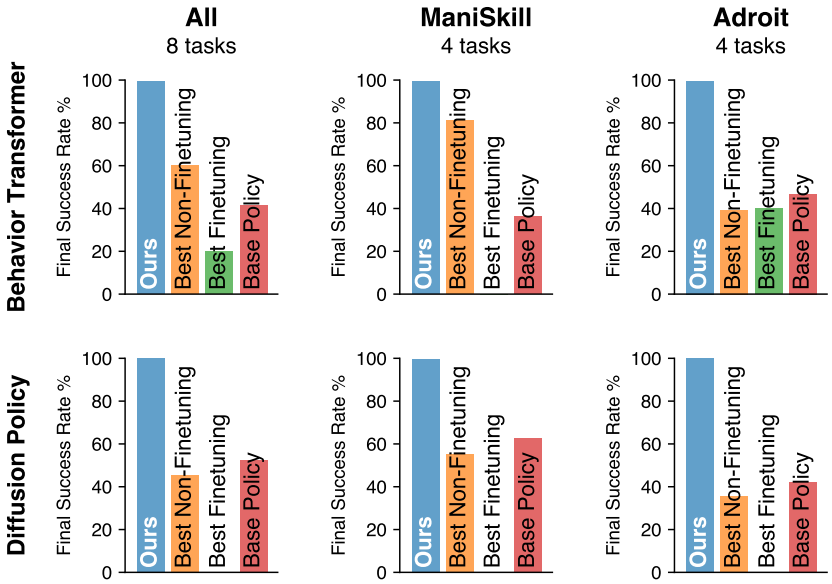

Figure 4.1. Policy Decorator improves base policy to near-perfect performance on two benchmarks, outperforming fine-tuning and non-fine-tuning baselines.

several challenges hinder this strategy. *Firstly*, many state-of-the-art imitation learning models have specific designs to accommodate the multimodal action distributions in the demonstrations, which unfortunately make them *non-trivial to fine-tune using RL*. For example, Behavior Transformer [182], MCNN [188], and VINN [146] all incorporate some non-differentiable components (clustering, nearest neighbor search) which are incompatible with the gradient-based optimization in RL. Similarly, Diffusion Policy [28] requires ground truth action labels to supervise its denoising process, but these action labels are unavailable in RL setups. *Secondly*, even if an imitation learning model were compatible with RL, the fine-tuning process would be prohibitively costly for two reasons: 1) the increasing number of parameters in modern large policy models, and 2) the extensive gradient updates required during sparse-reward RL training, a process known for its poor sample efficiency.

To devise a method for online improvement, we must first understand why an offline-trained imitation learning policy sometimes fails to solve tasks. As studied in [175, 173, 194, 26, 221], a major issue with policies learned from purely offline data is **compounding error**. Small errors gradually accumulate, eventually leading the policy to states not covered in the

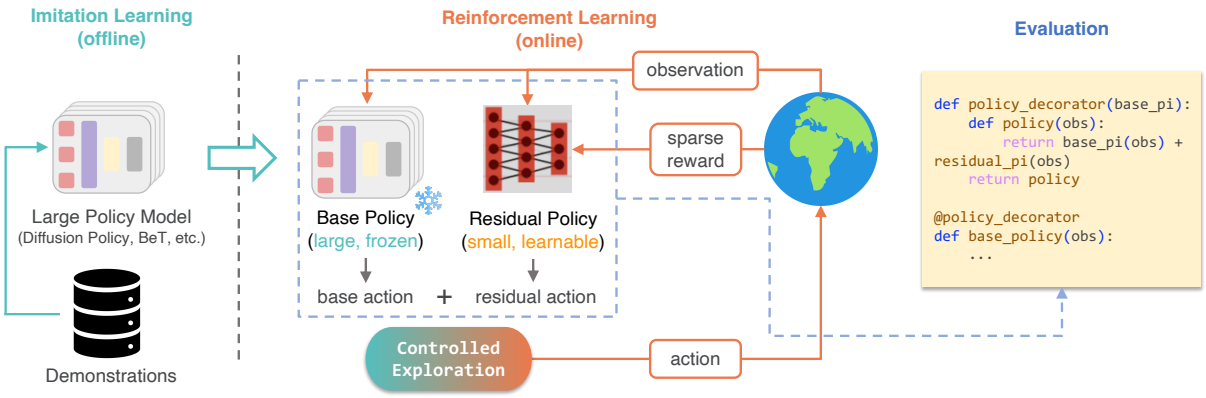

Figure 4.2. Our framework (Policy Decorator) improves large policy models through online interactions. We learn a residual policy via RL using controlled exploration strategies (Sec. 4.4.2). Once learned, it functions similarly to Python decorators—wrapping the base policy with an additional function to boost performance.

demonstration dataset. However, correcting these errors may only require minimal effort. Even if the final trajectory deviates significantly from the correct path, slight adjustments can bring it back on track, as shown in Fig. 4.3. In other words, the model only needs "refinement" for the finer parts of the tasks. Modeling such small adjustments typically does not necessitate complex architectures or large numbers of parameters. Therefore, we propose to online learn a **residual policy** (parameterized by a small network) to correct the behavior of the offline-trained imitation learning models, referred to as the "base policy" throughout this paper. This approach addresses the incompatibility between models and RL and avoids the costly gradient updates on large models.

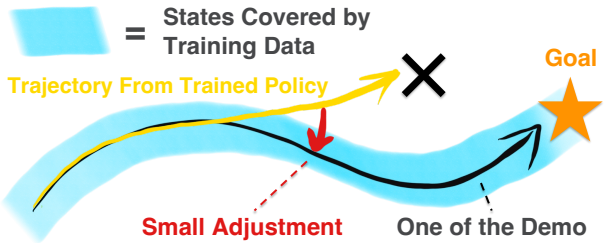

Figure 4.3. Small adjustments can bring deviated trajectories back on track.

While learning a residual policy through online RL [84, 7, 184, 231] can, in principle, refine a base policy, practical implementation is still challenging. As demonstrated in our

experiments (Sec. 4.5), without constraints, random exploration during RL training often leads to failure in tasks requiring precise control, resulting in no learning signals in sparse reward settings. To overcome these challenges and enable stable, sample-efficient learning, we propose a set of strategies to ensure the RL agent (with the residual policy) **explores the environment in a controlled manner**. This approach ensures that the agent continuously receives sufficient success signals while adequately exploring the environment. We call this framework the **Policy Decorator** because it functions similarly to decorators in Python—enhancing the original policy by wrapping it with an additional function to boost its performance, as illustrated in Fig. 4.2. Like Python decorators, our framework does not require any prior knowledge of the original policy and treats it as a black box, making it **model-agnostic**.

We evaluate our approach across a variety of benchmarks and imitation learning models. Specifically, we examine 8 tasks from 2 benchmarks: ManiSkill [132, 64] and Adroit [170], in conjunction with 2 state-of-the-art imitation learning models: Behavior Transformer [182] (action clustering + regression) and Diffusion Policy [28] (diffusion models + receding horizon control). Our results demonstrate that the Policy Decorator consistently improves various offline-trained large policy models to near-optimal performance in most cases. Furthermore, the learned policy maintains the desirable properties of the imitation learning policy, producing smooth motions rather than the jerky motions generated by pure RL policies.

To summarize, our contributions are as follows:

- Conceptually, we raise the critical research question: "How can large policy models be improved through online interactions?", and identify limitations of fine-tuning and vanilla residual RL.
- Technically, we propose Policy Decorator, a *model-agnostic* framework for refining large policy models through online environmental interactions.
- Empirically, we conduct extensive experiments on 8 challenging robotic tasks and 2 state-of-the-art imitation learning models, demonstrating Policy Decorator's advantages in both

task performance and learned policy properties.

4.2 Related Works

Learning from Demo Learning control policies through trial and error can be inefficient and unstable, prompting research into leveraging demonstrations to enhance online learning. Demonstrations can be utilized through pure offline imitation learning, including behavior cloning [154] and inverse reinforcement learning [142]. Alternatively, demonstrations can be incorporated during online learning, serving as off-policy experience [128, 70, 14, 137] or for on-policy regularization [90, 170]. Furthermore, demonstrations can be used to estimate reward functions for RL problems [218, 11, 209, 239, 186]. When the offline dataset includes both demonstrations and negative trajectories, offline-to-online RL approaches first apply offline RL to learn effective policy and value initializations from offline data, followed by online fine-tuning [136, 96, 119, 138]. *In this work, we adopt a more direct approach to utilize demonstrations: distilling demonstrations into a large policy model and subsequently improving it through online interactions.*

Residual Learning The concept of learning residual components has been widely applied across various domains, including addressing the vanishing gradient problem in deep neural networks [69, 206] and parameter-efficient fine-tuning [75]. In robotic control, researchers have employed online RL to learn corrective residual components for various base policies, such as hand-crafted controllers [84], non-parametric models [68], and pre-trained neural networks [7, 9]. Residual learning can also be achieved through supervised learning [82]. Our work focuses on the online improvement of *large policy models*, identifying residual policy learning as an ideal solution due to its model-agnostic nature. We highlight the *uncontrolled exploration issue in vanilla residual RL*, propose a set of strategies to address it, and further enhance its efficiency through careful examination of design choices.

Advanced Imitation Learning Imitation learning provides an effective approach to teaching robots complex skills. However, naive imitation learning often struggles to model

multi-modal distributions within demonstration datasets [28]. Several advanced methods have been proposed to address this limitation: [182, 101, 188, 146] represent actions as discrete values with offsets, [47] employs energy-based models, and [28] leverages diffusion models. While these techniques effectively learn from multi-modal data, they often create models that are *non-trivial to fine-tune using RL*. Even if they were compatible with RL, *the fine-tuning process can be computationally prohibitive* due to the large number of parameters in modern policy models. These limitations motivate our approach of using online residual policy learning to improve imitation learning models.

4.3 Problem Setup

In this paper, we focus on improving an offline-trained large policy (referred to as "base policy") through online interactions. We make the following assumptions:

1. An environment is available for online interactions with task success signals (sparse rewards).

2. The base policy may have a large number of parameters or complex architectures, making fine-tuning non-trivial or computationally expensive. This assumption holds for many modern large policy models [19, 18, 28, 182].

3. The base policy exhibits reasonable initial performance, though not perfect (i.e., it can make progress towards task completion, which is achievable by many state-of-the-art IL methods with a reasonable amount of demonstrations). An excessively poor base policy is not worth improving.

Note that our approach does not make any specific assumptions about model architectures or training methods. Instead, we treat these models as *black boxes* that take observations as input and produce actions as output. In our experiments, we choose to improve base policies trained by imitation learning rather than offline RL policies. This is because: 1) collecting demonstrations

alone is more cost-effective and thus more common; 2) as demonstrated in multiple studies [122, 47], imitation learning outperforms offline RL in demonstration-only settings.

4.4 Policy Decorator: Model-Agnostic Online Refinement

In this work, our goal is to online improve a large policy model, which is usually offline-trained by imitation learning and usually has some specific designs in model architecture. To this end, we propose *Policy Decorator*, a model-agnostic framework for refining large policy models via online interactions with environments. Fig. 4.2 provides an overview of our framework.

Policy Decorator is grounded on **learning a residual policy via reinforcement learning with sparse rewards**, which is described in Sec. 4.4.1. On top of it, we devise a set of strategies to ensure the RL agent (in combination with the base policy and the residual policy) explores the environment in a controlled manner. Such a **controlled exploration** mechanism is detailed in Sec. 4.4.2.

4.4.1 Learning Residual Policy via RL

Given the base policy π_{base}, we then train a residual policy π_{res} on top of it using reinforcement learning. The base policy π_{base} can be either deterministic (e.g., Behavior Transformer [182]) or stochastic (e.g., Diffusion Policy [28]), and it remains frozen during the RL process. The residual policy π_{res} is updated through RL gradients, so it should be a differentiable function compatible with RL gradients. In this work, we model the residual policy π_{res} as a Gaussian policy parameterized by a small neural network (either an MLP or a CNN, depending on the observation modality). To interact with the environment, the actions from both policies are combined by summing their output actions, i.e., the action executed in the environment is $\pi_{base}(s) + \pi_{res}(s)$. For stochastic policies, actions are sampled individually from both policies and then summed.

The residual policy is trained to maximize the expected discounted return derived from the sparse reward (i.e., the task's success signal). We employ the Soft Actor-Critic (SAC)

algorithm [65] due to its superior sample efficiency and stability.

4.4.2 Controlled Exploration

While learning a residual policy by RL can in principle refine a base policy, practical implementation can be challenging. As demonstrated in our experiments (Sec. 4.5), without constraints, random exploration during RL training often leads to failure in tasks requiring precise control, resulting in no learning signals in sparse reward settings. To overcome these challenges and enable stable, sample-efficient learning, we propose a set of strategies ensuring the RL agent (in combination with the base policy and the residual policy) **explores the environment in a controlled manner**. The goal is to make sure the agent continuously receives sufficient success signals while adequately exploring the environment.

Bounded Residual Action When using the residual policy to correct the base policy, we do not want the resulting trajectory to deviate too much from the original trajectory because it usually leads to failure. Instead, we expect the residual policy to only make a bit "refinement" at the finer parts of the tasks. To reflect this spirit, we bound the output of the residual policy within a certain scale. Since we use SAC as our backbone RL algorithm, the output of the policy is naturally bounded by a squashing function (`tanh`), whose range is $(-1, 1)$. We further scale the action sampled from the Gaussian policy with a hyperparameter α, making the range of the residual action $(-\alpha, \alpha)$. We found that an appropriate scale of residual action bound can be crucial for some precise tasks. We investigated the effects of hyperparameter α in Sec. 4.5.4.

Progressive Exploration Schedule Given that our residual policy is randomly initialized, the agent (combined with the base policy and residual policy) may exhibit highly random behavior and fail to succeed at the initial stage of learning. Therefore, the base policy alone, trained by imitation learning, can be more reliable during the early stages. As training progresses, the residual policy can be gradually improved, making it safer to incorporate its suggestions.

Inspired by the ε-greedy strategy used in DQN [128], we propose to progressively introduce actions from the residual policy into the agent's behavior policy. Specifically, the

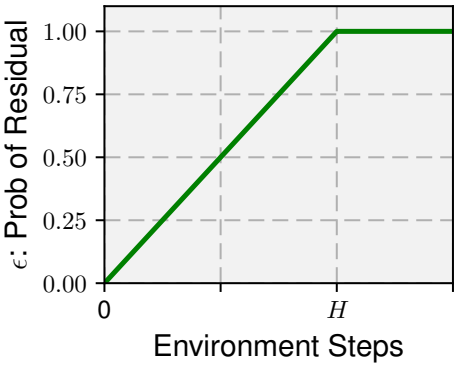

Figure 4.4. Progressive Exploration Schedule.

behavior policy will use actions from the residual policy to complement the base policy with probability ε and rely solely on the base policy with probability $1-\varepsilon$. Formally, during training,

$$\pi_{behavior}(s) = \begin{cases} \pi_{base}(s) + \pi_{res}(s) & \text{Uniform}(0,1) < \varepsilon \\ \pi_{base}(s) & \text{otherwise} \end{cases} \tag{4.1}$$

The parameter ε increases linearly from 0 to 1 over a specified number of time steps, as shown in Fig. 4.4, where H is a hyperparameter. Our experiments in Sec. 4.5.4 indicate that while tuning H can enhance sample efficiency, using a large H is generally a safe choice.

4.5 Experiments

The goal of our experimental evaluation is to study the following questions:

1. Can Policy Decorator **effectively refine offline-trained imitation policies using online RL with sparse rewards** under different setups (different tasks, base policy architectures, demonstration sources, and observation modalities)? (Sec. 4.5.3)

2. What are the **effects of the components** introduced by the Policy Decorator? (Sec. 4.5.4)

3. Does Policy Decorator generate **better task-solving behaviors** compared to other types of learning paradigms (e.g., pure IL and pure RL)? (Sec. 4.5.5)

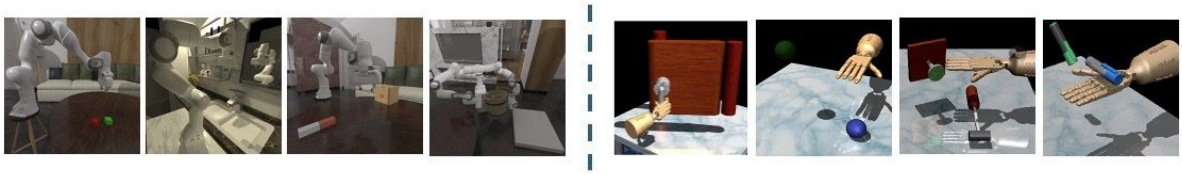

Figure 4.5. Tasks Visualizations. ManiSkill (left four figures) and Adroit (right four figures).

4.5.1 Experimental Setup

To validate Policy Decorator's versatility, our experimental setup incorporates *variations across the following dimensions*:

- **Task Types:** Stationary robot arm manipulation, mobile manipulation, dual-arm coordination, dexterous hand manipulation, articulated object manipulation, and high-precision tasks. Fig. 4.5 illustrates sample tasks from each benchmark.
- **Base Policies:** Behavior Transformer and Diffusion Policy.
- **Demo Sources:** Teleoperation, Task and Motion Planning, RL, and Model Predictive Control.
- **Observation Modalities:** State observation (low-dim) and visual observation (high-dim).

We summarize the key details of our setups as follows.

Task Description

Our experiments are conducted on 8 tasks across 2 benchmarks: ManiSkill (robotic manipulation; 4 tasks), and Adroit (dexterous manipulation; 4 tasks). See Fig. 4.5 for illustrations.

ManiSkill We consider four challenging tasks from ManiSkill. StackCube and PegInsertionSide demand *high-precision control*, with PegInsertion featuring a mere 3mm clearance. TurnFaucet and PushChair introduce *object variations*, where the base policy is trained on source environment objects, but target environments for online interactions contain different objects. These complexities make it **challenging for pure offline imitation learning to achieve**

near-perfect success rates, necessitating online learning approaches. For all ManiSkill tasks, we use 1000 demonstrations provided by the benchmark [132, 64] across all methods. These demonstrations are generated through task and motion planning, model predictive control, and reinforcement learning.

Adroit We consider all four dexterous manipulation tasks from Adroit: Door, Hammer, Pen, and Relocate. The tasks should be solved using a complex, 24-DoF manipulator, simulating a real hand. For all Adroit tasks, we use 25 demonstrations provided by the original paper [170] for all methods. These demonstrations are collected by human teleoperation.

Base Policy Model

We selected two popular imitation learning models as our base policy models for improvement.

Behavior Transformer [182] is a GPT-based policy architecture for behavior cloning. It handles multi-modal action distribution by representing an action as a combination of a cluster center (predicted by a classification head) and an offset (predicted by a regression head). The action cluster centers are determined by k means, which is non-differentiable, thus only the offset can be fine-tuned using RL gradients.

Diffusion Policy [28] is a state-of-the-art imitation learning method that leverages recent advancements in denoising diffusion probabilistic models. It generates robot action sequences through a conditional denoising diffusion process and employs action sequences with receding horizon control. The training of Diffusion Policy requires ground truth action labels to supervise its denoising process; however, these action labels are unavailable in RL setups, making the original training recipe incompatible with RL. Nevertheless, recent approaches have been developed to fine-tune diffusion models using RL in certain scenarios.

4.5.2 Baselines

We compare our approach against a set of strong baselines for online policy improvement, including both **fine-tuning-based methods** and **methods that do not involve fine-tuning**. A brief description of each baseline is provided below.

Fine-tuning Methods

As discussed in Sec. 4.1, making our base policies compatible with online RL is non-trivial. *We implemented several specific modifications to the base policies to enable fine-tuning*. Since we consider the problem of improving large policy models where full-parameter fine-tuning can be costly, we employ LoRA [75] for parameter-efficient fine-tuning.

Our fine-tuning baseline selection follows this rationale: we first choose a basic RL algorithm for each base policy based on their specific properties, which serves as a basic baseline. Additionally, assuming access to the demonstrations used to train the base policies, we consider various learning-from-demonstration methods as potential baselines. Table 4.1 lists the most relevant learning-from-demo baselines. From these, we select *the strongest and most representative methods in each category* and implement them on top of the basic RL algorithm we initially selected.

Basic RL We use SAC [65] as our basic fine-tuning method for Behavior Transformer, and use DIPO [224] for Diffusion Policy. For both methods, we initialize the actor with the pre-trained base policy and use a randomly initialized MLP for the critic.

RLPD [14] is a state-of-the-art online learning-from-demonstration method that *utilizes demonstrations as off-policy experience*. It enhances vanilla SACfd with critic layer normalization, symmetric sampling, and sample-efficient RL techniques.

ROT [67] is a representative online learning-from-demonstration algorithm that *utilizes demonstrations to derive dense rewards* and *for policy regularization*. It adaptively combines offline behavior cloning with online trajectory-matching based rewards.

Cal-QL [138] is a state-of-the-art *offline-to-online RL* method that "calibrates" the Q

Table 4.1. Potential Fine-tuning Baselines with Demos. We categorize potential learn-from-demo baselines into four distinct categories, and choose the best and most representative methods from each category as our main points of comparison. Selected baselines are in **bold**.

Category	Method
Demo for Reward Learning	**ROT** [67] GAIL [73] DAC [95]
Demo as Off-Policy Experience	**RLPD** [14] SACfd [208]
Demo as On-Policy Regularization	**ROT** [67] DAPG [170] AWAC [136]
Offline RL Online Fine-tuning	**Cal-QL** [138] IQL [96] CQL [99]

function in CQL [99] for efficient online fine-tuning. In our setting, we use the same demonstration set used in other baselines as the offline data for Cal-QL. Unlike other fine-tuning baselines that initialize the critic randomly, Cal-QL can *potentially benefit from the pre-trained critic*.

Non-fine-tuning Methods

JSRL [203] is a curriculum learning method that employs a guiding policy to bring the agent closer to the goal. In our setting, the pre-trained base policy serves as the guiding policy.

Residual RL [84] learns a residual control signal on top of a hand-crafted conventional controller. Unlike our approach, *it explores the environment in an entirely uncontrolled manner*. For a fair comparison, we replace its hand-crafted controller with our base policies.

FISH [68] builds upon Residual RL by incorporating a non-parametric VINN [146] policy and learning an online offset actor with optimal transport rewards.

4.5.3 Main Results & Analysis

Our Approach We evaluate Policy Decorator with Behavior Transformer and Diffusion Policy as base policies, and the results are summarized in Fig. 4.6 and 4.7, respectively (see Fig.

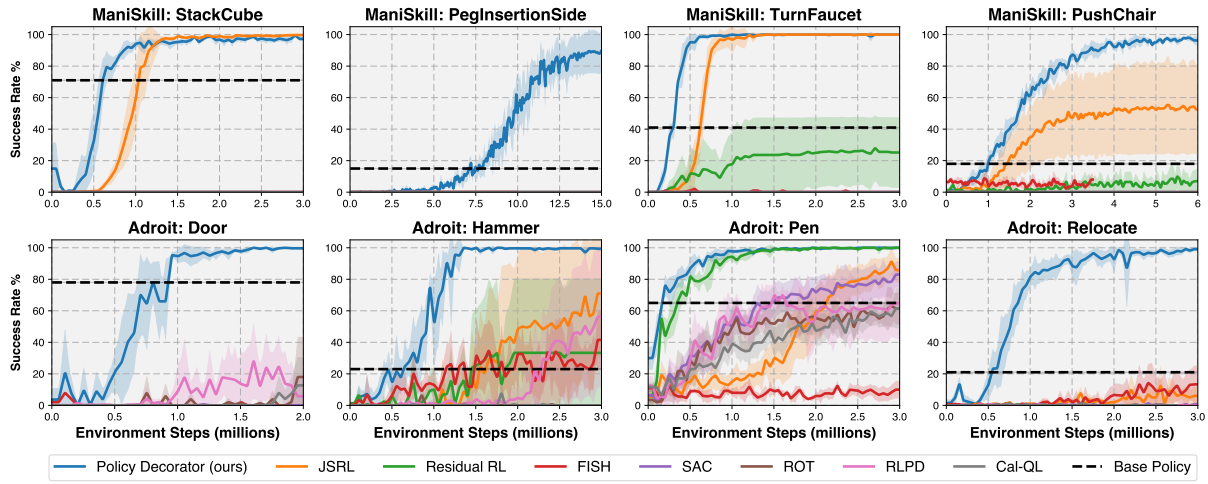

Figure 4.6. Results (with Behavior Transformer): During training, we evaluate the agent for 50 episodes every 50K environment steps. The curves depict the evaluation success rates averaged over ten seeds for our approach and three seeds for baselines. Shaded areas represent standard deviations. Our method consistently improves the base policy and outperforms all other baselines.

4.1 for a barplot). Policy Decorator improves the performance of both offline-trained policies to a near-perfect level on all tasks across ManiSkill and Adroit when given low-dimensional state observations. For Diffusion Policy, we did not test on StackCube and Door since the base policy already achieves near-optimal performance in these tasks.

Non-Finetuning Baselines Overall, JSRL performs the best among all baselines but only exceeds the base policy's performance on around half of the scenarios. Additionally, JSRL does not actually "improve" the base policy but instead learns an entirely new policy. This means that even if it achieves a high success rate, it does not preserve the desired properties of the original base policy, such as smooth and natural motion. Residual RL improves the base policy on 3 out of 6 tasks when combined with Diffusion Policy, but performs quite poorly when combined with Behavior Transformer. We suspect that this is because residual RL agents have a higher chance of obtaining task success signals through random exploration due to the stronger performance and robustness of the Diffusion Policy models. FISH performs poorly on most tasks, primarily due to the weak performance of the VINN.

Finetuning Baselines All fine-tuning-based baselines generally perform poorly in our

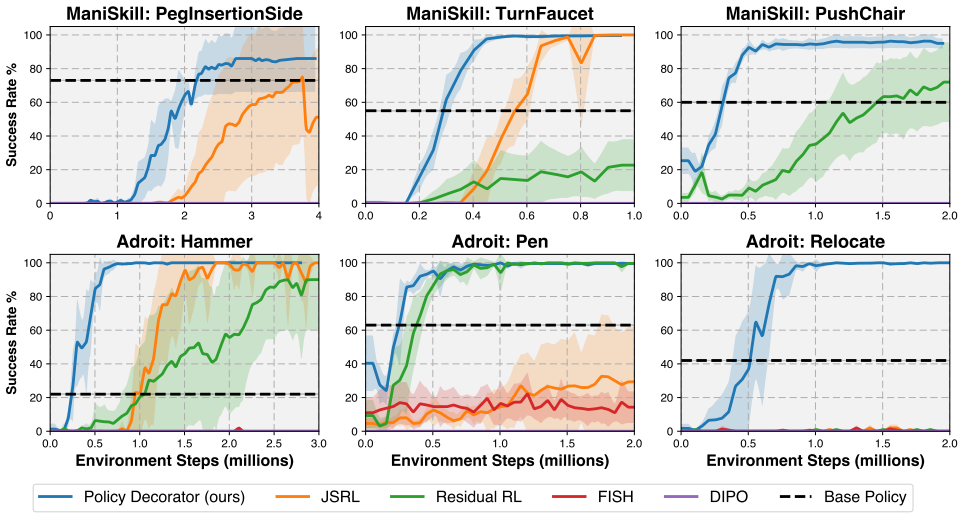

Figure 4.7. Results (with Diffusion Policy): The setup is similar to Fig. 4.6, but with different baselines due to the nature of Diffusion Policy. Since DIPO does not work at all on any tasks, we did not include other fine-tuning-based baselines built on top of DIPO. In addition, we did not test on StackCube and Adroit Door because the base policy is already near-optimal (99%+ success rates).

evaluation. Cal-QL and RLPD can improve Behavior Transformer on a few Adroit tasks but completely fail on ManiSkill tasks. We suspect this is because *the randomly initialized critic network cannot provide meaningful gradients* and quickly causes the agent to deviate significantly from the original trajectories. In contrast, our controlled exploration strategies help the agent remain exposed to success signals. While Cal-QL can theoretically learn a good critic from offline data, we found that the learned critic does not aid online fine-tuning when it is trained purely on demonstration data without negative trajectories. This degradation over the course of Cal-QL online training has also been observed by [223]. Another reason for the failure of fine-tuning-based methods is *the long-horizon nature of our tasks*. We observed that RL fine-tuning becomes effective when the task horizon is reduced. For Diffusion Policy, we observed that DIPO failed to obtain any success signals across all tasks, so we did not further test other fine-tuning-based methods that rely on DIPO as the backbone RL algorithm. We hypothesize that this failure is due to the receding horizon control in Diffusion Policy, which complicates the fine-tuning process. For instance, when Diffusion Policy predicts 16 actions but only the

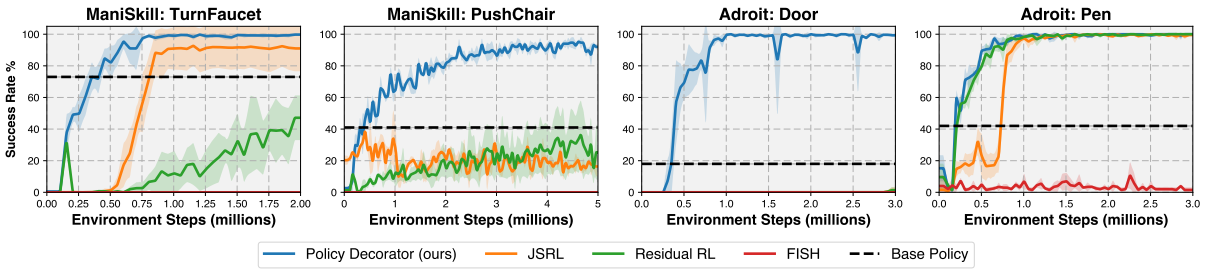

Figure 4.8. Results on Image Observations (with Diffusion Policy): Similar to Fig. 4.7, but using image observations instead of low-dimensional state observations. We selected two tasks with complex visual appearances from each benchmark.

first 8 are executed in the environment, *there is no clear method to supervise the latter 8 actions during fine-tuning*. Keeping the latter 8 actions unchanged is incorrect because once the first 8 actions are modified through fine-tuning, they may bring the agent to a new state where the latter 8 actions no longer apply.

Visual Observations Finally, we conducted experiments with visual observations. As shown in Fig. 4.8, the results validated that Policy Decorator also performs well with high-dim visual observations.

4.5.4 Ablation Study

We conducted various ablations on Stack Cube and Push Chair tasks to provide further insights.

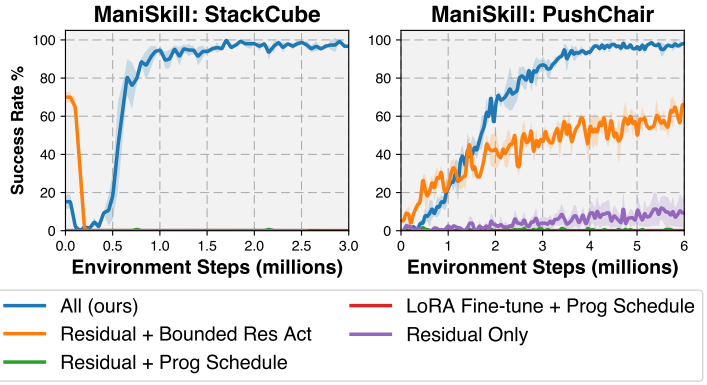

Figure 4.9. The importance of each component:1) residual policy learning; 2) progressive exploration schedule; and 3) bounded residual action.

Relative Importance of Each Component

We examined the relative importance of Policy Decorator's main components: 1) residual policy learning; 2) progressive exploration schedule; and 3) bounded residual action. We thoroughly evaluated *all possible combinations* of these components, with results shown in Fig. 4.9. Each component greatly contributes to the overall performance, both individually and collectively. While residual policy learning establishes the foundation of our framework, using it alone does not sufficiently improve the base policy. Bounded residual action is essential for effective residual policy learning, and the progressive exploration schedule further enhances sample efficiency.

Influence of Key Hyperparameters

Bound α of Residual Actions The hyperparameter α determines the maximum adjustment the residual policy can make. Fig. 4.10 illustrates how α affects the learning process. If α is too small, the final performance may be adversely affected. Conversely, if α is too large, it may lead to poor sample efficiency during training. Although certain values achieve optimal sample efficiency, α values within a broad range (e.g., 0.1 to 0.5 for PushChair and 0.03 to 0.1 for StackCube) eventually converge to similar success rates, albeit with varying sample efficiencies. This indicates that **while the choice of α is impactful, our method remains robust across a wide range of α values**. In practice, tuning α is relatively straightforward: **we typically set it close to the action scale observed in the demonstration dataset** and make minor adjustments as necessary.

***H* in Progressive Exploration Schedule** The hyperparameter H (see Fig. 4.4 for an illustration) controls the rate at which we switch from the base policy to the residual policy. From Fig. 4.11, we observe that a too-small H can lead to complete failure due to aggressive exploration, while a large H may result in relatively poor sample efficiency. Therefore, **tuning H can enhance sample efficiency** and ensure stable training. However, **using a large H is generally a safe choice** if sample efficiency is not the primary concern.

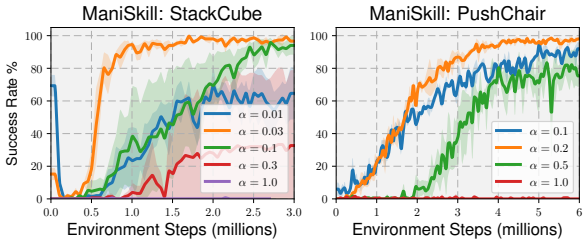

Figure 4.10. Different values of the bound α for Residual Actions.

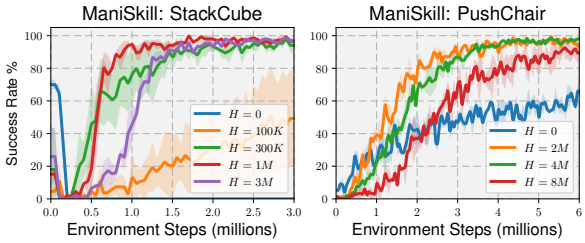

Figure 4.11. Different values of H in Progressive Exploration Schedule.

4.5.5 Properties of the Refined Policy

An intriguing aspect of Policy Decorator is its ability to **combine the strengths of both Imitation Learning and Reinforcement Learning policies**. Previous observations have highlighted that robotic policies trained solely by RL often exhibit jerky actions, rendering them unsuitable for real-world application [162]. Conversely, policies derived from demonstrations, whether from human teleoperation or motion planning, tend to produce more natural and smooth motions. However, the performance of such policies is constrained by the diversity and quantity of the demonstrations.

Our refined policy, learned through Policy Decorator, **achieves remarkably high success rates while retaining the favorable attributes of the base policy**. This is intuitive – by constraining residual actions, the resulting trajectory maintains proximity to the original trajectory, minimizing deviation.

Comparison with RL policies reveals that our refined approach exhibits significantly smoother behavior. Furthermore, when compared with offline-trained base policies, our refined policy demonstrates superior performance, effortlessly navigating through the finest part of the task.

4.6 Conclusions, Discussions, & Limitations

We propose the Policy Decorator framework, a flexible method for improving large behavior models using online interactions. We introduce controlled exploration strategies that

boost the base policy's performance efficiently. Our method achieves near-perfect success rates on most tasks while preserving the smooth motions typically seen in imitation learning models, unlike the jerky movements often found in reinforcement learning policies.

Limitations Enhancing large models with online interactions requires significant training time and resources. While learning a small residual policy reduces computational costs compared to fully fine-tuning the large model, the process remains resource-intensive, especially for slow-inference models like diffusion policies. We found that only a few critical states need adjustment. Future research could focus on identifying and correcting these points more precisely to improve efficiency.

4.7 Acknowledgment

This chapter, in full, is taken from "Policy Decorator: Model-Agnostic Online Refinement for Large Policy Model" by Xiu Yuan, Tongzhou Mu, Stone Tao, Yunaho Fang, Michael Zhang, Hao Su. It has been submitted for publication of the material as it may appear in the International Conference on Learning Representations (ICLR) 2025. The dissertation author was a primary investigator and author of this paper.

Chapter 5

Conclusions

This dissertation has presented a data-centric approach to robotics, motivated by the significant progress witnessed in other AI domains through large models trained on massive datasets. Recognizing the data bottleneck in Physical World AI, this work focused on three core components: environment, data, and policy. We introduced ManiSkill, a large-scale simulation benchmark for generalizable manipulation skill learning, to address the need for rich and diverse environments for data collection. Furthermore, we developed DrS, a method for learning reusable dense rewards, and Trajectory Translation, a framework for one-shot task generalization, to facilitate efficient demonstration acquisition for both short-horizon and long-horizon tasks. Finally, we introduced Policy Decorator, a model-agnostic online refinement method, enabling effective utilization of both offline demonstrations and online interactions to learn robust and generalizable policies.

These contributions collectively advance the field of robotics by providing tools and techniques for creating, collecting, and utilizing data more effectively. ManiSkill offers a valuable platform for benchmarking and developing generalizable manipulation skills. DrS simplifies the often tedious process of reward engineering, enabling more efficient reinforcement learning. Trajectory Translation facilitates one-shot learning for complex tasks, reducing the need for extensive task-specific training data. Policy Decorator offers a practical and efficient way to refine large policy models, bridging the gap between offline training and online adaptation.

Looking ahead, the pursuit of general-purpose robots remains a central challenge in robotics research. The paradigm of training large models on large datasets offers a promising path towards this goal. While the architecture of these models is likely to be transformer-based, the source of the necessary large-scale data remains an open question.

Several promising directions for scaling up robotic data collection warrant further exploration. One approach involves direct collection of large-scale robot data in the real world. This approach, while straightforward in concept, requires substantial engineering effort to build robust and scalable data collection pipelines, encompassing hardware, systems, and efficient teleoperation methods. A second direction focuses on collecting large-scale simulated robot data. This approach holds significant potential due to the scalability of simulation, but the sim-to-real gap poses a persistent challenge, requiring sophisticated sim-to-real transfer techniques. A third possibility involves leveraging large-scale non-robot data, such as YouTube videos, in conjunction with smaller-scale real-world robot data. This approach envisions pre-training world models on readily available non-robot data and subsequently using these models for planning and inferring robot actions through inverse dynamics models trained on smaller, targeted robot datasets.

While real-world robot data collection is currently receiving significant attention, the other two approaches—simulated data and leveraging non-robot data—offer compelling alternatives and deserve further investigation. A balanced approach, potentially combining these different data sources, may ultimately prove most effective in overcoming the data bottleneck and unlocking the full potential of large models for achieving general-purpose robots. The development of robust sim-to-real transfer methods, efficient techniques for extracting meaningful information from non-robot data, and innovative approaches for combining diverse data sources will be crucial for future progress in this exciting area of research.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[2] David Abel. A theory of abstraction in reinforcement learning. *arXiv preprint arXiv:2203.00397*, 2022.

[3] Karen E Adolph, Bennett I Bertenthal, Steven M Boker, Eugene C Goldfield, and Eleanor J Gibson. Learning in the development of infant locomotion. *Monographs of the society for research in child development*, pages i–162, 1997.

[4] Shailen Agrawal and Michiel van de Panne. Task-based locomotion. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016.

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

[6] 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.

[7] Minttu Alakuijala, Gabriel Dulac-Arnold, Julien Mairal, Jean Ponce, and Cordelia Schmid. Residual reinforcement learning from demonstrations. *arXiv preprint arXiv:2106.08050*, 2021.

[8] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5048–5058,

2017.

[9] Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. From imitation to refinement–residual rl for precise visual assembly. *arXiv preprint arXiv:2407.16677*, 2024.

[10] Dafni Antotsiou, Guillermo Garcia-Hernando, and Tae-Kyun Kim. Task-oriented hand motion retargeting for dexterous manipulation imitation. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[11] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando De Freitas. Playing hard exploration games by watching youtube. *Advances in neural information processing systems*, 31, 2018.

[12] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1726–1734. AAAI Press, 2017.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[14] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[15] Lionel Blondé and Alexandros Kalousis. Sample-efficient imitation learning via generative adversarial nets. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3138–3148. PMLR, 2019.

[16] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

[17] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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

[19] 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.

[20] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

[21] 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.

[22] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[23] Felix Burget, Armin Hornung, and Maren Bennewitz. Whole-body motion planning for manipulation of articulated objects. In *2013 IEEE International Conference on Robotics and Automation*, pages 1656–1662. IEEE, 2013.

[24] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*, 2019.

[25] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[26] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning*, pages 2058–2066. PMLR, 2015.

[27] 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.

[28] 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.

[29] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot

teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

[30] Sachin Chitta, Benjamin Cohen, and Maxim Likhachev. Planning for autonomous door opening with a mobile manipulator. In *2010 IEEE International Conference on Robotics and Automation*, pages 1799–1806. IEEE, 2010.

[31] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

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

[33] Open X-Embodiment Collaboration. Open X-Embodiment: Robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864, 2023.

[34] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[35] Daniela Corbetta. Perception, action, and intrinsic motivation in infants' motor-skill development. *Current Directions in Psychological Science*, 30(5):418–424, 2021.

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

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

[38] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

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

[40] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 30, 2017.

[41] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*,

2019.

[42] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell. Imitating latent policies from observation. In *International conference on machine learning*, pages 1755–1763. PMLR, 2019.

[43] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4497–4506, June 2021.

[44] Haoshu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11441–11450. IEEE, 2020.

[45] Kuan Fang, Yuke Zhu, Animesh Garg, Andrey Kurenkov, Viraj Mehta, Li Fei-Fei, and Silvio Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *The International Journal of Robotics Research*, 39(2-3):202–216, 2020.

[46] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017.

[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] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018.

[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] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[51] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. *Advances in neural information processing systems*, 31, 2018.

[52] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *CoRR*, abs/2106.06860, 2021.

[53] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[54] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[55] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2052–2062. PMLR, 2019.

[56] 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.

[57] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *arXiv preprint arXiv:2010.01083*, 2020.

[58] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020.

[59] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR, 2020.

[60] Robert Gieselmann and Florian T. Pokorny. Planning-augmented hierarchical reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):5097–5104, 2021.

[61] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[62] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

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

[64] 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.

[65] 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.

[66] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[67] Siddhant Haldar, Vaibhav Mathur, Denis Yarats, and Lerrel Pinto. Watch and match: Supercharging imitation with regularized optimal transport. In *Conference on Robot Learning*, pages 32–43. PMLR, 2023.

[68] Siddhant Haldar, Jyothish Pari, Anant Rai, and Lerrel Pinto. Teach a robot to fish: Versatile imitation from one minute of demonstrations. *arXiv preprint arXiv:2303.01497*, 2023.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[70] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[71] 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.

[72] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4565–4573, 2016.

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

[74] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*,

9(8):1735–1780, 1997.

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

[76] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[77] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

[78] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 540–550, 2020.

[79] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. *Advances in neural information processing systems*, 26, 2013.

[80] Stephen James, Michael Bloesch, and Andrew J Davison. Task-embedded control networks for few-shot imitation learning. *Conference on Robot Learning (CoRL)*, 2018.

[81] 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.

[82] Yunfan Jiang, Chen Wang, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Transic: Sim-to-real policy transfer by learning from online correction. *arXiv preprint arXiv:2405.10315*, 2024.

[83] Mingxuan Jing, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Chao Yang, Bin Fang, and Huaping Liu. Reinforcement learning from imperfect demonstrations under soft expert guidance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5109–5116, 2020.

[84] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[85] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in

belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[86] 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.

[87] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

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

[89] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International Conference on Machine Learning*, pages 2469–2478. PMLR, 2018.

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

[91] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[92] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118. IEEE, 2009.

[93] 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*, 2017.

[94] 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.

[95] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *arXiv preprint arXiv:1809.02925*, 2018.

[96] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[97] Aviral Kumar, Anikait Singh, Frederik Ebert, Mitsuhiko Nakamoto, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *arXiv preprint arXiv:2210.05178*, 2022.

[98] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[99] 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.

[100] Vikash Kumar, Abhishek Gupta, Emanuel Todorov, and Sergey Levine. Learning dexterous manipulation policies from experience and imitation. *arXiv preprint arXiv:1611.05095*, 2016.

[101] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

[102] Youngwoon Lee, Edward S. Hu, Zhengyu Yang, and Joseph J. Lim. To follow or not to follow: Selective imitation learning from observations. In *Conference on Robot Learning*, 2019.

[103] Youngwoon Lee, Joseph J Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. *arXiv preprint arXiv:2111.07999*, 2021.

[104] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2019.

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

[106] 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.

[107] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data

collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[108] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*, 2017.

[109] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2018.

[110] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.

[111] Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4051–4058. IEEE, 2020.

[112] Xiaolong Li, He Wang, Li Yi, Leonidas J. Guibas, A. Lynn Abbott, and Shuran Song. Category-level articulated object pose estimation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 3703–3712. IEEE, 2020.

[113] Zhijun Li, Ting Zhao, Fei Chen, Yingbai Hu, Chun-Yi Su, and Toshio Fukuda. Reinforcement learning of manipulation and grasping using dynamical movement primitives for a humanoidlike mobile manipulator. *IEEE/ASME Transactions on Mechatronics*, 23(1):121–131, 2017.

[114] Hongzhuo Liang, Xiaojian Ma, Shuang Li, Michael Görner, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang. PointNetGPD: Detecting grasp configurations from point sets. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[115] Yijiong Lin, John Lloyd, Alex Church, and Nathan F Lepora. Tactile gym 2.0: Sim-to-real deep reinforcement learning for comparing low-cost high-resolution robot touch. *IEEE Robotics and Automation Letters*, 7(4):10754–10761, 2022.

[116] Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. *arXiv preprint arXiv:1911.10947*, 2019.

[117] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[118] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[119] Jiafei Lyu, Xiaoteng Ma, Xiu Li, and Zongqing Lu. Mildly conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:1711–1724, 2022.

[120] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[121] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *2009 16th IEEE international conference on image processing (ICIP)*, pages 3501–3504. IEEE, 2009.

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

[123] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[124] Andrew Melnik, Luca Lach, Matthias Plappert, Timo Korthals, Robert Haschke, and Helge Ritter. Using tactile sensing to improve the sample efficiency and performance of deep deterministic policy gradients for simulated in-hand manipulation tasks. *Frontiers in Robotics and AI*, 8:538773, 2021.

[125] Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Scott Niekum, and Ufuk Topcu. Self-supervised online reward shaping in sparse-reward environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2369–2375. IEEE, 2021.

[126] Mayank Mittal, David Hoeller, Farbod Farshidian, Marco Hutter, and Animesh Garg. Articulated object interaction in unknown scenes with whole-body mobile manipulation. *arXiv preprint arXiv:2103.10534*, 2021.

[127] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

[128] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[129] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 909–918. Computer Vision Foundation / IEEE, 2019.

[130] Hans Moravec. Mind children: The future of robot and human intelligence. *Harvard UP*, 1988.

[131] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 2901–2910. IEEE, 2019.

[132] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[133] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[134] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3307–3317, 2018.

[135] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3307–3317, Red Hook, NY, USA, 2018. Curran Associates Inc.

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

[137] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[138] Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

[139] Venkatraman Narayanan and Maxim Likhachev. Task-oriented planning for manipulating articulated mechanisms under model uncertainty. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3095–3101. IEEE, 2015.

[140] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[141] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[142] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

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

[144] OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique Ponde de Oliveira Pinto, et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*, 2021.

[145] Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning? *Advances in Neural Information Processing Systems*, 34:14656–14668, 2021.

[146] Jyothish Pari, Nur Muhammad Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*, 2021.

[147] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pages 7487–7498. PMLR, 2020.

[148] 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.

[149] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 2050–2053, 2018.

[150] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

[151] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

[152] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, 2020.

[153] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J. Lim. Demonstration-guided reinforcement learning with learned skills. *5th Conference on Robot Learning*, 2021.

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

[155] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8494–8502. IEEE Computer Society, 2018.

[156] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.

[157] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[158] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9277–9286, 2019.

[159] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep

learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017.

[160] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[161] Yuzhe Qin, Rui Chen, Hao Zhu, Meng Song, Jing Xu, and Hao Su. S4g: Amodal single-view single-shot se (3) grasp detection in cluttered scenes. In *Conference on robot learning*, pages 53–65. PMLR, 2020.

[162] 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.

[163] Yuzhe Qin, Wei Yang, Binghao Huang, Karl Van Wyk, Hao Su, Xiaolong Wang, Yu-Wei Chao, and Dieter Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. *arXiv preprint arXiv:2307.04577*, 2023.

[164] Zengyi Qin, Kuan Fang, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Keto: Learning keypoint representations for tool manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7278–7285. IEEE, 2020.

[165] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[166] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[167] Ilija Radosavovic, Xiaolong Wang, Lerrel Pinto, and Jitendra Malik. State-only imitation learning for dexterous manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7865–7871. IEEE, 2020.

[168] 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.

[169] 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.

[170] 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.

[171] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006.

[172] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[173] 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.

[174] 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.

[175] 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, 2011.

[176] Manolis Savva, Jitendra Malik, Devi Parikh, Dhruv Batra, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, and Vladlen Koltun. Habitat: A platform for embodied AI research. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9338–9346. IEEE, 2019.

[177] M Saylor and P Ganea. *Active learning from infancy to childhood*. Springer, 2018.

[178] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[180] Levine Sergey, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA*, pages 26–30, 2015.

[181] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016.

[182] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[183] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchapmi, et al. igibson 1.0: a simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7520–7527. IEEE, 2021.

[184] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[185] Avi Singh, Eric Jang, Alexander Irpan, Daniel Kappler, Murtaza Dalal, Sergey Levinev, Mohi Khansari, and Chelsea Finn. Scalable multi-task imitation learning with autonomous improvement. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2167–2173. IEEE, 2020.

[186] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[187] Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine. Avid: Learning multi-stage tasks via pixel-level translation of human videos. *arXiv preprint arXiv:1912.04443*, 2019.

[188] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, James Weimer, and Insup Lee. Memory-consistent neural networks for imitation learning. *arXiv preprint arXiv:2310.06171*, 2023.

[189] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. BEHAVIOR: benchmark for everyday household activities in virtual, interactive, and ecological environments. *CoRR*, abs/2108.03332, 2021.

[190] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.

[191] Wael Suleiman, Eiichi Yoshida, Fumio Kanehiro, Jean-Paul Laumond, and André Monin.

On human motion imitation by humanoid robot. In *2008 IEEE International conference on robotics and automation*, pages 2697–2704. IEEE, 2008.

[192] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

[193] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[194] Umar Syed and Robert E Schapire. A reduction from apprenticeship learning to classification. *Advances in neural information processing systems*, 23, 2010.

[195] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

[196] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[197] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473, 2017.

[198] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *2010 IEEE International Conference on Robotics and Automation*, pages 2397–2403. IEEE, 2010.

[199] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

[200] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[201] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[202] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. *Advances in*

Neural Information Processing Systems, 32, 2019.

[203] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. Jump-start reinforcement learning. In *International Conference on Machine Learning*, pages 34556–34583. PMLR, 2023.

[204] 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.

[205] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[206] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[207] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[208] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[209] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *2019 international conference on robotics and automation (ICRA)*, pages 754–760. IEEE, 2019.

[210] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3540–3549.

PMLR, 2017.

[211] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[212] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018.

[213] Zheng Wu, Wenzhao Lian, Vaibhav Unhelkar, Masayoshi Tomizuka, and Stefan Schaal. Learning dense rewards for contact-rich manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6214–6221. IEEE, 2021.

[214] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020.

[215] Fei Xia, Amir Roshan Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9068–9079. IEEE Computer Society, 2018.

[216] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11094–11104. IEEE, 2020.

[217] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[218] Annie Xie, Avi Singh, Sergey Levine, and Chelsea Finn. Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning*, pages 40–52. PMLR, 2018.

[219] Danfei Xu and Misha Denil. Positive-unlabeled reward learning. In *Conference on Robot Learning*, pages 205–219. PMLR, 2021.

[220] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, B. Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In

Thirty-ninth International Conference on Machine Learning, 2022.

[221] Tian Xu, Ziniu Li, and Yang Yu. Error bounds of imitating policies and environments. *Advances in Neural Information Processing Systems*, 33:15737–15749, 2020.

[222] Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. Deep view synthesis from sparse photometric images. *ACM Transactions on Graphics (ToG)*, 38(4):1–13, 2019.

[223] Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning. *arXiv preprint arXiv:2310.15145*, 2023.

[224] Long Yang, Zhixiong Huang, Fenghao Lei, Yucun Zhong, Yiming Yang, Cong Fang, Shiting Wen, Binbin Zhou, and Zhouchen Lin. Policy representation via diffusion probability model for reinforcement learning. *arXiv preprint arXiv:2305.13122*, 2023.

[225] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

[226] 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.

[227] 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.

[228] 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.

[229] 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. *arXiv preprint arXiv:2010.14406*, 2020.

[230] Vicky Zeng, Timothy E Lee, Jacky Liang, and Oliver Kroemer. Visual identification of articulated object parts. *arXiv preprint arXiv:2012.00284*, 2020.

[231] Shangtong Zhang, Wendelin Boehmer, and Shimon Whiteson. Deep residual reinforcement learning. *arXiv preprint arXiv:1905.01072*, 2019.

[232] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. *arXiv preprint arXiv:2012.09164*, 2020.

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

[234] Allan Zhou, Eric Jang, Daniel Kappler, Alex Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine, and Chelsea Finn. Watch, try, learn: Meta-learning from demonstrations and reward. *arXiv preprint arXiv:1906.03352*, 2019.

[235] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[236] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

[237] Zhuangdi Zhu, Kaixiang Lin, Bo Dai, and Jiayu Zhou. Learning sparse rewarded tasks from sub-optimal demonstrations. *arXiv preprint arXiv:2004.00530*, 2020.

[238] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[239] Konrad Zolna, Alexander Novikov, Ksenia Konyushkova, Caglar Gulcehre, Ziyu Wang, Yusuf Aytar, Misha Denil, Nando de Freitas, and Scott Reed. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.