UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning Generalist Robot Manipulation Policies

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

in

Computer Science

by

Jiayuan Gu

Committee in charge:

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

2024

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

University of California San Diego

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF LISTINGS

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Hao Su for his invaluable guidance and support as the chair of my dissertation committee. I am also profoundly thankful to Professor Henrik Christensen, Professor Zhuowen Tu, and Professor Xiaolong Wang for their insightful contributions and unwavering support as my committee members.

I am grateful to my collaborators, including my labmates, colleagues and friends across various institutions, such as Google DeepMind, Meta AI, Waymo, and Uber ATG, for their camaraderie, inspiration, and shared wisdom that significantly enriched my research journey. In particular, I wish to express my profound gratitude to Fanbo, Yuzhe and Tongzhou, whose assistance was crucial in my transition from Computer Vision to Embodied AI, a challenging yet rewarding endeavor. Additionally, I am immensely grateful to Ted, Quan, Sean, Chelsea, and Karol for their insightful guidance during my internships at Google DeepMind, which has been instrumental in my development. My sincere thanks also go to Devendra and Jitendra at Meta AI for their patient and long-term guidance. Moreover, the support from Raquel, Wei-chiu, Sivabalan, and Wenyuan at Uber ATG, as well as Charles at Waymo, was pivotal to my achievements. Their contributions were not just helpful but essential to my success.

Thanks to Professor Jianbo Shi, for his generous support and invaluable advice, which have been instrumental in shaping my academic and professional paths.

Thanks to my parents and my grandmother, whose unwavering and unconditional support has been my guiding light through every challenging moment. Their enduring love and encouragement have been the foundation upon which I've built my resilience, enabling me to navigate the darkest times with hope and perseverance.

Chapter 2, in full, is a reprint of the material published in the 2023 International Conference on Learning Representations (ICLR): "ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills" (Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, Hao Su). The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material published in the 2024 International Conference on Learning Representations (ICLR): "RT-Trajectory: Robotic Task Generalization via Hindsight Trajectory Sketches" (Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, Ted Xiao). The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of the material published in the 2023 International Conference on Learning Representations (ICLR): "Multi-skill Mobile Manipulation for Object Rearrangement" (Jiayuan Gu, Devendra Singh Chaplot, Hao Su, Jitendra Malik). The dissertation author was the primary investigator and author of this paper.

VITA

2014-2018 Bachelor of Science in Intelligence Science and Technology, Peking University

2018-2024 Doctor of Philosophy in Computer Science, University of California San Diego

PUBLICATIONS

[1] Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. In *The Eleventh International Conference on Learning Representations*, 2024

[2] Xiaoshuai Zhang, Rui Chen, Ang Li, Fanbo Xiang, Yuzhe Qin, Jiayuan Gu, Z. Ling, Minghua Liu, Peiyu Zeng, Songfang Han, Zhiao Huang, Tongzhou Mu, Jing Xu, and Hao Su. Close the optical sensing domain gap by physics-grounded active stereo sensor simulation. *IEEE Transactions on Robotics*, 39:2429–2447, 2022

[3] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang 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 *The Eleventh International Conference on Learning Representations*, 2023

[4] Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill mobile manipulation for object rearrangement. In *The Eleventh International Conference on Learning Representations*, 2023

[5] Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020

[6] Tongzhou Mu, Jiayuan Gu, Zhiwei Jia, Hao Tang, and Hao Su. Refactoring policy for compositional generalizability using self-supervised object proposals. *Advances in Neural Information Processing Systems*, 33:8883–8894, 2020

[7] Wei-Chiu Ma, Shenlong Wang, Jiayuan Gu, Sivabalan Manivasagam, Antonio Torralba, and Raquel Urtasun. Deep feedback inverse problem solver. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 229–246. Springer, 2020

[8] Jiayuan Gu, Wei-Chiu Ma, Sivabalan Manivasagam, Wenyuan Zeng, Zihao Wang, Yuwen Xiong, Hao Su, and Raquel Urtasun. Weakly-supervised 3d shape completion in the wild. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 283–299. Springer, 2020

[9] Maximilian Jaritz, Jiayuan Gu, and Hao Su. Multi-view pointnet for 3d scene understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019

[10] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. *Advances in neural information processing systems*, 31, 2018

[11] Jiayuan Gu, Han Hu, Liwei Wang, Yichen Wei, and Jifeng Dai. Learning region features for object detection. In *Proceedings of the european conference on computer vision (ECCV)*, pages 381–395, 2018

[12] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3588–3597. IEEE, 2018

ABSTRACT OF THE DISSERTATION

Learning Generalist Robot Manipulation Policies

by

Jiayuan Gu

Doctor of Philosophy in Computer Science

University of California San Diego, 2024

Professor Hao Su, Chair

The pursuit of Artificial General Intelligence necessitates intelligent agents with a "body" to interact with and learn from their environments, central to the goal of Embodied AI. Despite remarkable success in learning specialized skills for individual tasks through data-driven approaches, learning generalist robot manipulation policies, which master generalizable skills for a wide range of tasks, remains challenging. In this dissertation, we present our efforts to develop scalable simulation systems and explore effective representations that facilitate the learning of generalist robot policies.

One major challenge is the high cost and inefficiency of collecting high-quality, diverse demonstration data in the real world. Simulations, serving as proxies for the real world, are more

affordable and accessible, allowing us to scale up demonstration collection and policy evaluation more easily. To this end, we develop *ManiSkill2*, a simulation benchmark for generalizable manipulation skills. This platform features over 2000 objects and 4 million demonstration frames for 20 out-the-box task families. We also provide a wide range of baselines and host a public leaderboard for the community to evaluate object-level generalization on manipulation skills.

Crucial to making full use of available demonstration data is the development of suitable representations, enabling robots to adapt to a broad spectrum of tasks. We propose *RT-Trajectory*, which explores enhancing task-level generalization by leveraging existing demonstration datasets with a novel policy conditioning: coarse trajectory sketch. This sketch outlines the desired motion of the robot's end-effector, empowering the policy to adapt to unseen tasks with novel semantics and movements in a promptable way.

Moreover, in *Multi-skill Mobile Manipulation (M3)*, we study a modular approach to tackle long-horizon mobile manipulation tasks, which decomposes a full task into a sequence of subtasks solved by chaining multiple manipulation and navigation skills. We demonstrate how subtask definitions significantly shape skill quality and utility in the context of skill chaining. Accordingly, we redefine stationary manipulation and point-goal navigation skills into more versatile mobile manipulation and region-goal navigation skills.

Chapter 1

Introduction

1.1 Generalist Robot Manipulation Policies

The pursuit of Artificial General Intelligence (AGI) that can match or surpass human capabilities requires the development of intelligent agents equipped with a "body" to interact with and learn from environments. This foundational concept underpins Embodied AI, which aims to to create agents, such as robots, adept at performing complex, interactive tasks. While there has been significant progress in teaching robots specialized skills for specific tasks through data-driven methods, the challenge of developing generalist robot manipulation policies capable of mastering a wide array of skills remains.

Before delving deeper, it is essential to define key concepts such as "generalist robot policies", "skills", and "tasks" as used in this dissertation. Note that the definitions of these concepts can be subjective and vary across different communities. Ideally, a generalist robot policy can achieve a wide range of tasks in varied environments like human. In reinforcement learning literature, a *task* can be formulated as a Markov decision process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ of state space $\mathcal{S}$, action space $\mathcal{A}$, transition distribution $P(s'|s, a)$, reward function $R(s, a, s')$, reward discount factor γ, and associated with its initial state distribution ρ_0. More precisely, MDPs encountered in the real world are typically partially observed (POMDP). For simplicity, we use the general MDP framework here. A policy is a function $\pi(s) : \mathcal{S} \mapsto \mathcal{A}$ mapping from state space to action space. A task can be solved by finding the optimal policy π^*

that maximizes the expected return $E_\pi \left[\sum_{t=0} \gamma^t R_t(s_t, a_t, s_{t+1})\right]$. The goal of a task is implicitly defined by its reward function.

The term "task" frequently appears in policy learning literature but is often used with varying degrees of specificity. The scope of a task can range widely, from manipulating a specific object within a specific setting (e.g., picking up a cube on a tabletop) to handling a category of objects across diverse environments (e.g. picking up generic objects from different receptacles in various rooms). This variability in task granularity typically correlates with the size of the state space involved. Conversely, the objective of a task (e.g., "pick up") tends to be more straightforward to define. The term "skill" is commonly used to denote a set of tasks sharing the same objective. Note that "skill" is sometimes used interchangeably with the policy that accomplishes these tasks. Given the concepts of *task*, *policy*, and *skill*, a *generalist robot policy* refers to a policy that enables a robot to perform diverse skills to solve a wide array of tasks in various environments. Here, *environments* usually refer to the states generally irrelevant to tasks, e.g., background. Besides, a *skill* that is capable of manipulating a wide array of objects, including those previously unseen, in various configurations (e.g., initial poses of objects, initial setups of the robot, and background) can be termed a *generalizable skill*.

Historically, robotics research focused on addressing specific tasks within controlled settings. Classical approaches typically demand carefully engineered models of the environment and the robot. For instance, optimal control methods [124, 123] depend on good dynamic models. Motion planning [139] usually requires full knowledge of the environment, under the assumption that tasks are quasi-static. Task planning [32] relies on predefined object classes and action primitives. Despite the generality of these approaches, the complexity of accurately modeling both the environment and the robot limits their applicability to specific tasks and settings.

In contrast, learning-based methods train manipulation policies in an end-to-end fashion that predict control outputs directly from sensory inputs, thereby bypassing the need for explicit model engineering. However, many prior works often struggle to generalize across different tasks and environments. Reinforcement learning (RL) methods learn policies through trial and error,

showing success in a range of real-world tasks such as bin picking [56] and PCB insertion [68]. Despite these successes, most real-world RL implementations necessitate specialized hardware and algorithms for automatic environment resetting and success evaluation. Moreover, they tend to be sample-inefficient and may pose safety risks, hindering their scalability. Imitation learning (IL) approaches, which teach the robot to imitate expert demonstrations, have achieved notable successes in dexterous manipulation [18, 138] within real-world settings. However, their applicability remains constrained to specific environments [130], largely due to the scarcity of high-quality demonstration data and the challenges associated with scaling data collection efforts.

Foundation models [7] provide a promising solution to achieve superior generalization that generalist robot policies demand. A foundation model is characterized as "any model that is trained on broad data (generally using self-supervision at scale) that can be transferred or adapted (e.g., fine-tuned) to a wide range of downstream tasks". To date, the most successful foundation models have emerged from the fields of Computer Vision (CV) and Natural Language Processing (NLP), including Vision Foundation Models [86, 59], Large Language Models [13], and Vision-Language Models [94, 2]. More recently, there has been an emergence of foundation models specifically tailored for robotics applications, known as "robotics foundation models". These models, such as Visual-Language-Action (VLA) models [11], are trained on both internet-scale datasets and extensive collections of robot trajectories, offering enhanced generalizability, such as the ability to adapt to previously unseen concepts. This development is in line with the concept of generalist robot policies, aiming to significantly broaden the applicability and effectiveness of robotic systems.

1.2 Challenges

The acquisition of large-scale, high-quality and diverse data is essential for learning generalist robot manipulation policies. The common practice is to collect data from human demonstration through crowd-sourced teleoperation [12, 77], which is expensive and time-

consuming. For example, it took over 17 months to collect over 130K episodes for the RT-1 dataset [12] with a fleet of 13 robots. While recent works such as ALOHA [138] and GELLO [125] offer cost-effective teleoperation frameworks, collecting demonstrations at a scale similar to the internet-scale image and text data remains daunting, particularly across a broad spectrum of locations and robot embodiments. In addition, real-world data collection poses significant safety concerns.

To address these challenges, many studies have explored generating data in simulated environments. Simulations allow for classical methods like task and motion planing [22] or model predicative control [82], which require accurate models, to generate extensive demonstrations. Simulations also support domain randomization [111, 87] for training policies transferable from simulation to reality. While there has been notable success in training sim-to-real policies for tasks like dexterous manipulation [92, 137], the development of policies capable of addressing extensive visual diversity has not yet reached the same level. This shortfall is attributed to the limited diversity and realism of objects and scenes within simulated environments, highlighting an area in need of further innovation and exploration.

A key attribute of generalist robot policies is their versatility. A generalist robot policy can be prompted to execute specific tasks, essentially being goal-conditioned. The literature studies various representations of policy conditioning, including languages [50, 12], goal images [9], and videos [50]. Large language models have demonstrated the significant role that prompting can play. Yet, the influence of different policy conditioning methods on the generalizability of a policy remains an open question.

Furthermore, the criteria for determining a suitable skill (or goal) are rarely discussed. Humans usually break down daily chores into several achievable goals, addressing them in sequence. A common strategy in robotics involves integrating high-level planners with low-level skills, as exemplified in [1]. Even with identical goals, different formulations of a skill can significantly affect the whole system. For example, performing a task such as opening a door is more feasible with whole-body movement for a mobile manipulator than with arm movement

alone. Whole-body movement can also mitigate imperfect navigation by enabling the robot to adjust its position to find the optimal stance for task execution. This also indicates that prompting generalist robot policies to execute suitable skills is crucial for the robustness of robotics systems.

1.3 Overview of Techniques and Contributions

To address the aforementioned challenges, we introduce one piece of work (Chapter 2) focused on developing scalable simulation systems that facilitate the learning of generalizable manipulation skills. Additionally, we discuss two methods regarding task specification (Chapter 3) and skill formulation (Chapter 4), aimed at improving generalizability and robustness of robotics systems.

1.3.1 Building Simulated Environments for Generalizable Manipulation Skills

Simulations, as proxies for the real world, are more affordable and accessible for data collection and policy evaluation. Although a wide range of simulated benchmarks [133, 49, 140, 75, 24, 109, 82, 104, 63, 34, 90] have been established, there still remains a lack of well-suited datasets and benchmarks to assess policy generalizability, particularly at the object level. To this end, we have developed a simulation benchmark, ManiSkill2 [40], tailored for generalizable manipulation skills. This platform features over 2000 objects and 4 million demonstration frames for 20 skills. We provide a wide range of baselines (sense-plan-act, RL, IL) and maintain an online evaluation system, enabling the community to benchmark distinct algorithms. Moreover, it highlights an asynchronous RPC-based render server-client system designed to optimize throughput and GPU memory usage. We manage to collect samples with an RGBD-input PPO policy at about 2000 FPS 1 with 1 GPU and 16 CPU processors on a regular workstation, doubling the performance of prior works [109, 75].

Notably, our benchmark is based on fully simulated dynamics, instead of simplifying grasping behaviors [109, 24]. Simulating physically realistic grasping behavior is pivotal for

examining object-level generalizability, as the diverse topologies and geometries of objects significantly influence how a robot can grasp and manipulate them. Consequently, it poses significant challenges in demonstration collection. To tackle this, we employ a hybrid approach combining task and motion planning (TAMP), model predictive control (MPC), and reinforcement learning (RL), to collect demonstrations in a scalable way. For each task (object), we either apply TAMP if the task is basically quasi-static manipulation, or design shaped reward functions to search or train a specialist agent through MPC or RL if the task involves rich contacts or underactuated systems.

1.3.2 Conditioning Policies on Trajectory Sketches for Robotic Task Generalization

Language-conditioned policies like RT-1 [12] struggle to generalize to new scenarios that require extrapolation of language specifications even if similar motions are seen during training. Our key insight is that this kind of generalization becomes feasible if we represent the task through rough trajectory sketches that indicate desired end-effector motions and interactions. We propose a policy conditioning method, RT-Trajectory [38] using such rough trajectory sketches, that is practical, easy to specify, and enhances the policy's ability to execute unseen tasks. We find that trajectory sketches strike a balance between being detailed enough to express low-level motion-centric guidance while being coarse enough to allow the learned policy to interpret the trajectory sketch in the context of situational visual observations. Moreover, trajectory sketches serve as a versatile interface for communicating with robot policies, allowing for specifications through human inputs like drawings or videos, or automated techniques such as text-to-image generation. Our method is able to perform a wider range of unseen tasks in the real world, compared to language-conditioned and goal-conditioned policies, when provided the same training data.

1.3.3 Improving Skill Formulations for Robust Skill Chaining

In [36], we study how to formulate skills that can be chained to solve long-horizon mobile manipulation tasks. Prior works (e.g., [109]) chain multiple stationary manipulation skills with point-goal navigation, which are learned individually on subtasks. This framework suffers from compounding errors in skill chaining. For example, navigating to an improper location can lead to the unrecoverable failure for succeeding stationary manipulation skills. Thus, we propose that the manipulation skills should include mobility to have flexibility in interacting with the target object from multiple locations and at the same time the navigation skill could have multiple end points which lead to successful manipulation. We operationalize these concepts by implementing mobile manipulation skills and region-goal navigation reward. Our approach, evaluated on three complex long-horizon tasks in the Home Assistant Benchmark [109], has achieved the SOTA performance and won 1st place at the Habitat Rearrangement Challenge 2022.

1.4 Additional Work Done During my Doctoral Career

During my doctoral studies, I have delved into a variety of topics within Computer Vision and Machine Learning, focusing particularly on 3D understanding and reconstruction, as well as on policy learning paradigms.

In our work [39], we confronted the complex issue of 3D shape completion from unaligned and real-world partial point clouds, which are often sparse, noisy, and misaligned. We developed a weakly-supervised methodology that concurrently estimates the 3D canonical shape and 6D pose for alignment, leveraging multi-view geometry constraints. This innovative approach not only deduces complete shapes from single partial point clouds but also facilitates the registration of partial point clouds, showing encouraging outcomes on both synthetic and real datasets without the need for explicit shape and pose labels.

Our study in [51] tackled the integration of 2D image and 3D point cloud data to enhance 3D scene comprehension. The proposed MVPNet aggregates features from multi-view images

into 3D point clouds and uses a point-based network for merging these features in 3D space. This significantly boosts 3D semantic segmentation performance on the ScanNetV2 benchmark, illustrating the benefits of merging dense image features with sparse point cloud data and offering valuable insights for future fusion methodologies.

In [81], we investigated how to engineer a policy with the capability for compositional generalizability. We introduced a two-stage framework that refines a high-reward teacher policy into a generalizable student policy, employing an object-centric GNN-based design with self-supervised learning for object recognition from images. This method outperformed baselines on challenging tasks that demand compositional generalizability. The specialist-generalist framework proposed in this research, as further detailed in [40], demonstrates our approach to collecting demonstrations. Initially, we train specialist RL agents tailored to individual environments with minimal variation. A generalist model with increased capacity is then trained to integrate demonstrations collected by the specialist agents.

These research endeavors have built a strong interdisciplinary foundation and breadth of knowledge, enriching my contributions to the field of embodied AI.

Chapter 2

Building Simulated Environments for Generalizable Manipulation Skills

Generalizable manipulation skills, which can be composed to tackle long-horizon and complex daily chores, are one of the cornerstones of Embodied AI. However, existing benchmarks, mostly composed of a suite of simulatable environments, are insufficient to push cutting-edge research works because they lack object-level topological and geometric variations, are not based on fully dynamic simulation, or are short of native support for multiple types of manipulation tasks. To this end, we present ManiSkill2, the next generation of the SAPIEN ManiSkill benchmark, to address critical pain points often encountered by researchers when using benchmarks for generalizable manipulation skills. ManiSkill2 includes 20 manipulation task families with 2000+ object models and 4M+ demonstration frames, which cover stationary/mobile-base, single/dual-arm, and rigid/soft-body manipulation tasks with 2D/3D-input data simulated by fully dynamic engines. It defines a unified interface and evaluation protocol to support a wide range of algorithms (e.g., classic sense-plan-act, RL, IL), visual observations (point cloud, RGBD), and controllers (e.g., action type and parameterization). Moreover, it empowers fast visual input learning algorithms so that a CNN-based policy can collect samples at about 2000 FPS with 1 GPU and 16 processes on a regular workstation. It implements a render server infrastructure to allow sharing rendering resources across all environments, thereby significantly reducing memory usage. We open-source all codes of our benchmark (simulator, environments, and

baselines) and host an online challenge open to interdisciplinary researchers.

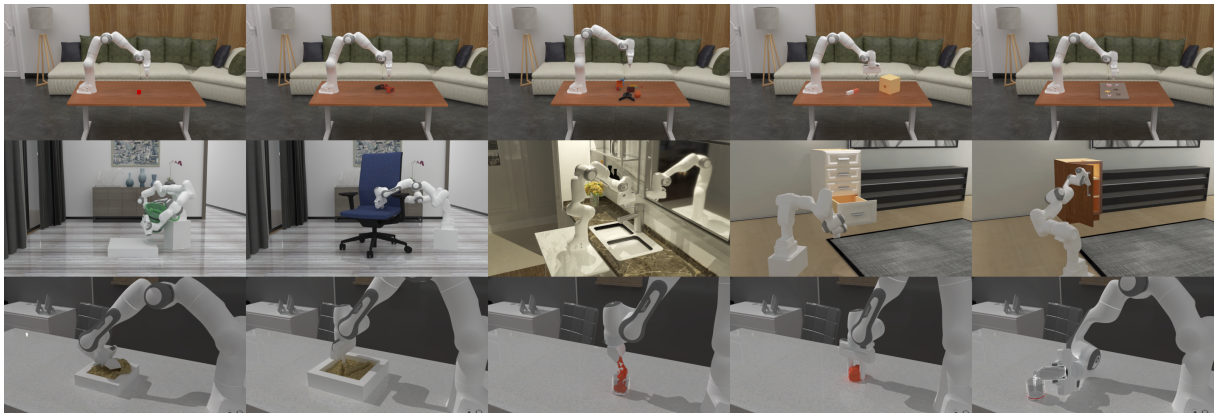


Figure 2.1. ManiSkill2 provides a unified, fast, and accessible system that encompasses well-curated manipulation tasks (e.g., stationary/mobile-base, single/dual-arm, rigid/soft-body).

2.1 Introduction

Mastering human-like manipulation skills is a fundamental but challenging problem in Embodied AI, which is at the nexus of vision, learning, and robotics. Remarkably, once humans have learnt to manipulate a category of objects, they are able to manipulate unseen objects (e.g., with different appearances and geometries) of the same category in unseen configurations (e.g., initial poses). We refer such abilities to interact with a great variety of even unseen objects in different configurations as **generalizable manipulation skills**. Generalizable manipulation skills are one of the cornerstones of Embodied AI, which can be composed to tackle long-horizon and complex daily chores [1, 36]. To foster further interdisciplinary and reproducible research on generalizable manipulation skills, it is crucial to build a versatile and public benchmark that focuses on object-level topological and geometric variations as well as practical manipulation challenges.

However, most prior benchmarks are insufficient to support and evaluate progress in learning generalizable manipulation skills. In this work, we present **ManiSkill2**, the next generation of SAPIEN ManiSkill Benchmark [82], which extends upon fully simulated dynamics,

a large variety of articulated objects, and large-scale demonstrations from the previous version. Moreover, we introduce significant improvements and novel functionalities, as shown below.

1) A Unified Benchmark for Generic and Generalizable Manipulation Skills: There does not exist a standard benchmark to measure different algorithms for generic and generalizable manipulation skills. It is largely due to well-known challenges to build realistically simulated environments with diverse assets. Many benchmarks bypass critical challenges as a trade-off, by either adopting abstract grasp [24, 109, 104] or including few object-level variations [140, 133, 49]. Thus, researchers usually have to make extra efforts to customize environments due to limited functionalities, which in turn makes reproducible comparison difficult. For example, [103] modified 18 tasks in [49] to enable few variations in initial states. Besides, some benchmarks are biased towards a single type of manipulation, e.g., 4-DoF manipulation in [133]. To address such pain points, ManiSkill2 includes a total of 20 verified and challenging manipulation task families of multiple types (stationary/mobile-base, single/dual-arm, rigid/soft-body), with over 2000 objects and 4M demonstration frames, to support generic and generalizable manipulation skills. All the tasks are implemented in a unified OpenAI Gym [10] interface with fully-simulated dynamic interaction, supporting multiple observation modes (point cloud, RGBD, privileged state) and multiple controllers. A unified protocol is defined to evaluate a wide range of algorithms (e.g., sense-plan-act, reinforcement and imtation learning) on both seen and unseen assets as well as configurations. In particular, we implement a cloud-based evaluation system to publicly and fairly compare different approaches.

2) Real-time Soft-body Environments: When operating in the real world, robots face not only rigid bodies, but many types of soft bodies, such as cloth, water, and soil. Many simulators have supported robotic manipulation with soft body simulation. For example, MuJoCo [112] and Bullet [21] use the finite element method (FEM) to enable the simulation of rope, cloth, and elastic objects. However, FEM-based methods cannot handle large deformation and topological changes, such as scooping flour or cutting dough. Other environments, like SoftGym [65] and ThreeDWorld [30], are based on Nvidia Flex, which can simulate large deformations, but

cannot realistically simulate elasto-plastic material, e.g., clay. PlasticineLab [48] deploys the continuum-mechanics-based material point method (MPM), but it lacks the ability to couple with rigid robots, and its simulation and rendering performance have much room for improvement. We have implemented a custom GPU MPM simulator from scratch using Nvidia's Warp [72] JIT framework and native CUDA for high efficiency and customizability. We further extend Warp's functionality to support more efficient host-device communication. Moreover, we have supported a 2-way dynamics coupling interface that enables any rigid-body simulation framework to interact with the soft bodies, allowing robots and assets in ManiSkill2 to interact with soft-body simulation seamlessly. To our knowledge, ManiSkill2 is the first embodied AI environment to support 2-way coupled rigid-MPM simulation, and also the first to support real-time simulation and rendering of MPM material.

3) Multi-controller Support and Conversion of Demonstration Action Spaces: Controllers transform policies' action outputs into motor commands that actuate the robot, which define the action space of a task. [78, 140] show that the choice of action space has considerable effects on exploration, robustness and sim2real transferability of RL policies. For example, task-space controllers are widely used for typical pick-and-place tasks, but might be suboptimal compared to joint-space controllers when collision avoidance [109] is required. ManiSkill2 supports a wide variety of controllers, e.g., joint-space controllers for motion planning and task-space controllers for teleoperation. A flexible system is also implemented to combine different controllers for different robot components. For instance, it is easy to specify a velocity controller for the base, a task-space position controller for the arm, and a joint-space position controller for the gripper. It differs from [140], which only supports setting a holistic controller for all the components. Most importantly, ManiSkill2 embraces a unique functionality to convert the action space of demonstrations to a desired one. It enables us to exploit large-scale demonstrations generated by any approach regardless of controllers.

4) Fast Visual RL Experiment Support: Visual RL training demands millions of samples from interaction, which makes performance optimization an important aspect in environment

design. Isaac Gym [75] implements a fully GPU-based vectorized simulator, but it lacks an efficient renderer. It also suffers from reduced usability (e.g., difficult to add diverse assets) and functionality (e.g., object contacts are inaccessible). EnvPool [121] batches environments by a thread pool to minimize synchronization and improve CPU utilization. Yet its environments need to be implemented in C++, which hinders fast prototyping (e.g., customizing observations and rewards). As a good trade-off between efficiency and customizability, our environments are fully scripted in Python and vectorized by multiple processes. We implement an asynchronous RPC-based render server-client system to optimize throughput and reduce GPU memory usage. We manage to collect samples with an RGBD-input PPO policy at about 2000 FPS [1] with 1 GPU and 16 CPU processors on a regular workstation.

2.2 Building Environments for Generalizable Manipulation Skills

Building high-quality environments demands cross-disciplinary knowledge and expertise, including physical simulation, rendering, robotics, machine learning, software engineering, *etc*. Our workflow highlights a verification-driven iterative development process, which is illustrated in Sec. 2.6.1. Different approaches, including task and motion planning (TAMP), model predictive control (MPC) and reinforcement learning (RL), can be used to generate demonstrations according to characteristics and difficulty of tasks, which verify environments as a byproduct.

2.2.1 Heterogeneous Task Families

ManiSkill2 embraces a heterogeneous collection of 20 task families. A task family represents a family of task variants that share the same objective but are associated with different assets and initial states. For simplicity, we interchangeably use *task* short for *task family*. Distinct types of manipulation tasks are covered: rigid/soft-body, stationary/mobile-base, single/dual-arm.

[1] The FPS is reported for rigid-body environments.

In this section, we briefly describe 4 groups of tasks. More details can be found in Sec. 2.8.

Soft-body Manipulation

ManiSkill2 implements 6 soft-body manipulation tasks that require agents to move or deform soft bodies into specified goal states through interaction.

1) *Fill*: filling clay from a bucket into the target beaker;

2) *Hang*: hanging a noodle on the target rod;

3) *Excavate*: scooping up a specific amount of clay and lifting it to a target height;

4) *Pour*: pouring water from a bottle into the target beaker. The final liquid level should match the red line on the beaker.

5) *Pinch*: deforming plasticine from an initial shape into a target shape; target shapes are generated by randomly pinching initial shapes, and are given as RGBD images or point clouds from 4 views.

6) *Write*: writing a target character on clay. Target characters are given as 2D depth maps.

A key challenge of these tasks is to reason how actions influence soft bodies, e.g., estimating displacement quantity or deformations, which will be illustrated in Sec 2.5.2.

Precise Peg-in-hole Assembly

Peg-in-hole assembly is a representative robotic task involving rich contact. We include a curriculum of peg-in-hole assembly tasks that require an agent to place an object into its corresponding slot. Compared to other existing assembly tasks, ours come with two noticeable improvements. First, our tasks target high precision (small clearance) at the level of millimeters, as most day-to-day assembly tasks demand. Second, our tasks emphasize the contact-rich insertion process, instead of solely measuring position or rotation errors.

1) *PegInsertionSide*: a single peg-in-hole assembly task inspired by MetaWorld [133]. It involves an agent picking up a cuboid-shaped peg and inserting it into a hole with a clearance of 3mm on the box. Our task is successful only if half of the peg is inserted, while the counterparts in prior works only require the peg head to approach the surface of the hole.

2) *PlugCharger*: a dual peg-in-hole assembly task inspired by RLBench [49], which involves an agent picking up and plugging a charger into a vertical receptacle. Our assets (the charger and holes on the receptacle) are modeled with realistic sizes, allowing a clearance of 0.5mm, while the counterparts in prior works only examine the proximity of the charger to a predefined position without modeling holes at all.

3) *AssemblingKits*: inspired by Transporter Networks [135], this task involves an agent picking up and inserting a shape into its corresponding slot on a board with 5 slots in total. We devise a programmatic way to carve slots on boards given any specified clearance (e.g., 0.8mm), such that we can generate any number of physically-realistic boards with slots. Note that slots in environments of prior works are visual marks, and there are in fact no holes on boards.

We generate demonstrations for the above tasks through TAMP. This demonstrates that it is feasible to build precise peg-in-hole assembly tasks solvable in simulation environments, without abstractions from prior works.

Stationary 6-DoF Pick-and-place

6-DoF pick-and-place is a widely studied topic in robotics. At the core is grasp pose [74, 93, 108]. In ManiSkill2, we provide a curriculum of pick-and-place tasks, which all require an agent to pick up an object and move it to a goal specified as a 3D position. The diverse topology and geometric variations among objects call for generalizable grasp pose predictions.

1) *PickCube*: picking up a cube and placing it at a specified goal position;

2) *StackCube*: picking up a cube and placing it on top of another cube. Unlike PickCube, the goal placing position is not explicitly given; instead, it needs to be inferred from observations.

3) *PickSingleYCB*: picking and placing an object from YCB [14];

4) *PickSingleEGAD*: picking and placing an object from EGAD [80];

5) *PickClutterYCB*: The task is similar to *PickSingleYCB*, but multiple objects are present in a single scene. The target object is specified by a visible 3D point on its surface.

Our pick-and-place tasks are deliberately designed to be challenging. For example,

the goal position is randomly selected within a large workspace ($30 \times 50 \times 50\, cm^3$). It poses challenges to sense-plan-act pipelines that do not take kinematic constraints into consideration when scoring grasp poses, as certain high-quality grasp poses might not be achievable by the robot.

Mobile/Stationary Manipulation of Articulated Objects

We inherit four mobile manipulation tasks from ManiSkill1 [82], which are *PushChair*, *MoveBucket*, *OpenCabinetDoor*, and *OpenCabinetDrawer*. We also add a stationary manipulation task, *TurnFaucet*, which uses a stationary arm to turn on faucets of various geometries and topology (details in Sec. 2.8.3).

Besides the above tasks, we have one last task, *AvoidObstacles*, which tests the navigation ability of a stationary arm to avoid a dense collection of obstacles in space while actively sensing the scene.

2.2.2 Multi-Controller Support and Conversion of Demonstration Action Spaces

The selection of controllers determines the action space. ManiSkill2 supports multiple controllers, e.g., *joint position*, *delta joint position*, *delta end-effector pose*, *etc*. Unless otherwise specified, controllers in ManiSkill2 translate input actions, which are desired configurations (e.g., joint positions or end-effector pose), to joint torques that drive corresponding joint motors to achieve desired actions. For instance, input actions to *joint position*, *delta joint position*, and *delta end-effector pose* controllers are, respectively, desired absolute joint positions, desired joint positions relative to current joint positions, and desired $\mathbb{SE}(3)$ transformations relative to the current end-effector pose. See Sec. 2.7 for a full description of all supported controllers.

Demonstrations are generated with one controller, with an associated action space. However, researchers may select an action space that conforms to a task but is different from the original one. Thus, to exploit large-scale demonstrations, it is crucial to convert the original action space to many different target action spaces while reproducing the kinematic and dynamic

processes in demonstrations. Let us consider a pair of environments: a source environment with a *joint position* controller used to generate demonstrations through TAMP, and a target environment with a *delta end-effector pose* controller for Imitation / Reinforcement Learning applications. The objective is to convert the source action $a_{\text{src}}(t)$ at each timestep t to the target action $a_{\text{tgt}}(t)$. By definition, the target action (*delta end-effector pose*) is $a_{\text{tgt}}(t) = \bar{T}_{\text{tgt}}(t) \cdot T_{\text{tgt}}^{-1}(t)$, where $\bar{T}_{\text{tgt}}(t)$ and $T_{\text{tgt}}(t)$ are respectively desired and current end-effector poses in the target environment. To achieve the same dynamic process in the source environment, we need to match $\bar{T}_{\text{tgt}}(t)$ with $\bar{T}_{\text{src}}(t)$, where $\bar{T}_{\text{src}}(t)$ is the desired end-effector pose in the source environment. $\bar{T}_{\text{src}}(t)$ can be computed from the desired joint positions ($a_{\text{src}}(t)$ in this example) through forward kinematics ($FK(\cdot)$). Thus, we have

$$a_{\text{tgt}}(t) = \bar{T}_{\text{tgt}}(t) \cdot T_{\text{tgt}}^{-1}(t) = \bar{T}_{\text{src}}(t) \cdot T_{\text{tgt}}^{-1}(t) = FK(\bar{a}_{\text{src}}(t)) \cdot T_{\text{tgt}}^{-1}(t)$$

Note that our method is closed-loop, as we instantiate a target environment to acquire $T_{\text{tgt}}^{-1}(t)$. For comparison, an open-loop method would use $T_{\text{src}}^{-1}(t)$ and suffers from accumulated execution errors.

2.3 Real-time Soft Body Simulation and Rendering

In this section, we describe our new physical simulator for soft bodies and their dynamic interactions with the existing rigid-body simulation. Our key contributions include: 1) a highly efficient GPU MPM simulator; 2) an effective 2-way dynamic coupling method to support interactions between our soft-body simulator and any rigid-body simulator, in our case, SAPIEN. These features enable us to create the first real-time MPM-based soft-body manipulation environment with 2-way coupling.

Rigid-soft Coupling

Our MPM solver is MLS-MPM [45] similar to PlasticineLab [48], but with a different contact modeling approach, which enables 2-way coupling with external rigid-body simulators. The coupling works by transferring rigid-body poses to the soft-body simulator and transferring soft-body forces to the rigid-body simulator. At the beginning of the simulation, all collision shapes in the external rigid-body simulator are copied to the soft-body simulator. Primitive shapes (box, capsule, etc.) are represented by analytical signed distance functions (SDFs); meshes are converted to SDF volumes, stored as 3D CUDA textures. After each rigid-body simulation step, we copy the poses and velocities of the rigid bodies to the soft-body simulator. During soft-body simulation steps, we evaluate the SDF functions at MPM particle positions and apply penalty forces to the particles, accumulating forces and torques for rigid bodies. In contrast, PlasticineLab evaluates SDF functions on MPM grid nodes, and applies forces to MPM grids. Our simulator supports both methods, but we find applying particle forces produces fewer artifacts such as penetration. After soft-body steps, we read the accumulated forces and torques from the soft-body simulator and apply them to the external rigid-body simulator. This procedure is summarized in Sec. 2.9.1. Despite being a simple 2-way coupling method, it is very flexible, allowing coupling with any rigid-body simulator, so we can introduce anything from a rigid-body simulator into the soft-body environment, including robots and controllers.

Performance Optimization

The performance of our soft-body simulator is optimized in 4 aspects. First, the simulator is implemented in Warp, Nvidia's JIT framework to translate Python code to native C++ and CUDA. Therefore, our simulator enjoys performance comparable to C and CUDA. Second, we have optimized the data transfer (e.g., poses and forces in the 2-way coupling) between CPU and GPU by further extending and optimizing the Warp framework; such data transfers are performance bottlenecks in other JIT frameworks such as Taichi [46], on which PlasticineLab is based. Third, our environments have much shorter compilation time and startup time compared

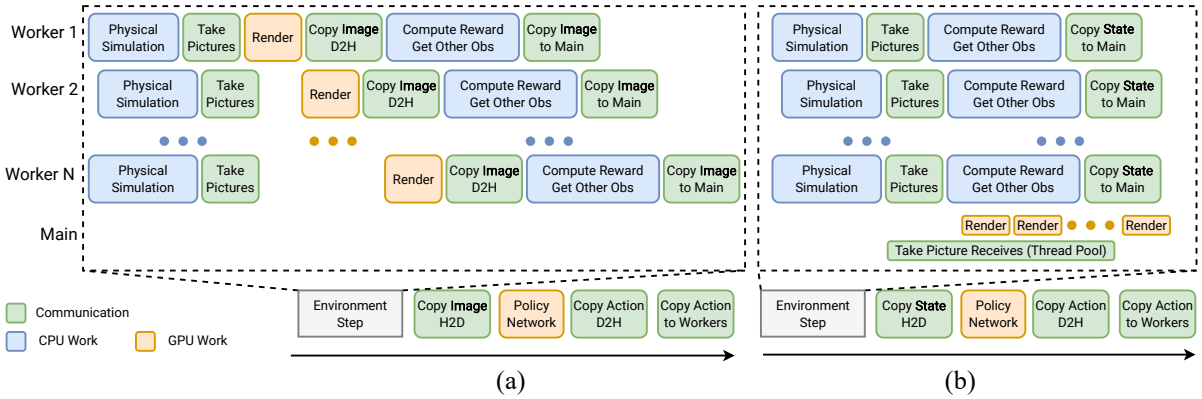


Figure 2.2. Two pipelines for visual RL sample collection. (a) Sequential pipeline. (b) Our pipeline with asynchronous rendering and render server improves CPU utilization, reduces data transfer, and saves memory.

to PlasticineLab thanks to the proper use of Warp compiler and caching mechanisms. Finally, since the simulator is designed for visual learning environments, we also implement a fast surface rendering algorithm for MPM particles, detailed in Sec. 2.9.2. Detailed simulation parameters and performance are provided in Sec. 2.9.3.

2.4 Parallelizing Physical Simulation and Rendering

ManiSkill2 aims to be a general and user-friendly framework, with low barriers for customization and minimal limitations. Therefore, we choose Python as our scripting language to model environments, and the open-source, highly flexible SAPIEN [128] as our physical simulator. The Python language determines that the only way to effectively parallelize execution is through multi-process, which is integrated in common visual RL training libraries [95, 120]. Under the multi-process environment paradigm, we make engineering efforts to enable our environments to surpass previous frameworks by increased throughput and reduced GPU memory usage.

What are the Goals in Visual-Learning Environment Optimization? The main goal of performance optimization in a visual-learning environment is to **maximize total throughput**: the number of samples collected from all (parallel) environments, measured in steps (frames) per second. Another goal is to **reduce GPU memory usage**. Typical Python-based multi-process

environments are wasteful in GPU memory usage, since each process has to maintain a full copy of GPU resources, e.g., meshes and textures. Given a fixed GPU memory budget, less GPU memory spent on rendering resources means more memory can be allocated for neural networks.

Asynchronous Rendering and Server-based Renderer: Fig. 2.2(a) illustrates a typical pipeline (sequential simulation and rendering) to collect samples from multi-process environments. It includes the following stages: (1) do physical simulation on worker processes; (2) take pictures (update renderer GPU states and submit draw calls); (3) wait for GPU render to finish; (4) copy image observations to CPU; (5) compute rewards and get other observations (e.g., robot proprioceptive info); (6) copy images to the main python process and synchronize; (7) copy these images to GPU for policy learning; (8) forward the policy network on GPU; (9) copy output actions from the GPU to the simulation worker processes.

We observe that the CPU is idling during GPU rendering (stage 3), while reward computation (stage 5) often does not rely on rendering results. Thus, we can increase CPU utilization by starting reward computation immediately after stage 2. We refer to this technique as **asynchronous rendering**.

Another observation is that images are copied from GPU to CPU on each process, passed to the main python process, and uploaded to the GPU again. It would be ideal if we can keep the data on GPU at all times. Our solution is to use a **render server**, which starts a thread pool on the main Python process and executes rendering requests from simulation worker processes, summarized in figure 2.2(b). The render server eliminates GPU-CPU-GPU image copies, thereby reducing data transfer time. It allows GPU resources to share across any number of environments, thereby significantly reducing memory usage. It enables communication over network, thereby having the potential to simulate and render on multiple machines. It requires minimal API changes – the only change to the original code base is to receive images from the render server. It also has additional benefits in software development with Nvidia GPU, which we detail in Sec. 2.10.2.

Comparison: We compare the throughput of ManiSkill2 with 3 other framework that

Table 2.1. Comparison of sample collection speed (FPS) on *PickCube* across different frameworks.

	ManiSkill2 Server	ManiSkill2 Sync	Habitat 2.0	RoboSuite 1.3	Isaac Gym
Total FPS (rand. action)	**2487±24**	942±19	1275±10	924±3	865±35
Total FPS (nature CNN)	**2532±63**	931±4	1224±13	894±15	835±5
Optimal #Envs	64	32	64	32	512

Table 2.2. Comparison of GPU memory usage between ManiSkill2 and Habitat 2.0. GPU memory usage on multi-process environments with 74 YCB objects each.

#Envs	ManiSkill2 Server	Habitat 2.0
4	4.9G	6.4G
8	5.1G	12.9G
64	5.8G	(OOM)

support visual RL: Habitat 2.0 [109], RoboSuite 1.3 [140], and Isaac Gym [75]. We build a *PickCube* environment in all simulators. We use similar physical simulation parameters (500Hz simulation frequency and 20Hz control frequency[2]), and we use the GPU pipeline for rendering. Images have resolution 128x128. All frameworks are given a computation budget of 16 CPU cores (logical processors) of an Intel i9-9960X CPU with 128G memory and 1 RTX Titan GPU with 24G memory. We test with random actions and also with a randomly initialized nature CNN [79] as policy network. We test all frameworks on 16, 32, 64, 128, 256, 512 parallel environments and report the highest performance. Results are shown in Table 2.1 (more details in Sec. 2.10.3). An interesting observation is that our environment performs the best when the number of parallel environments exceeds the number of CPU cores. We conjecture that this is because execution on CPU and GPU are naturally efficiently interleaved through OS or driver-level schedulers when the requested computation far exceeds the available resources.

To complement results in Table 2.1, we plot further details about the relationship between the number of parallel environments and the policy sample collection speed across different

[2]A fair comparison of different frameworks is still challenging as their simulation and rendering differ in fidelity. We try our best to match the simulation parameters among the frameworks.

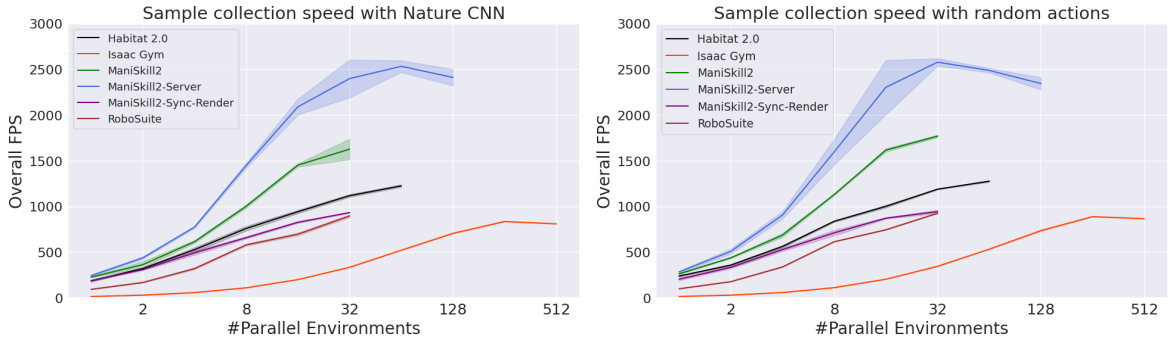


Figure 2.3. Comparison of sample collection speed (FPS) with random actions and with Nature CNN-sampled actions across different frameworks and different numbers of parallel environments. "ManiSkill2-Sync-Render" refers to ManiSkill2 with synchronous rendering and without render server. "ManiSkill2" refers to ManiSkill2 with asynchronous rendering but without render server. "ManiSkill2-Server" refers to ManiSkill2 with both asynchronous rendering and render server enabled. Some curves are not fully drawn beyond a certain number of parallel environments due to performance drop ("ManiSkill2-Server" & "Isaac Gym"), GPU out of memory ("RoboSuite" & "ManiSkill2-Sync-Render" & "ManiSkill2"), crash ("Habitat 2.0").

frameworks in Figure 2.3. We adopt an agent that outputs random actions, along with a CNN-based agent that uses a randomly-initialized nature CNN [79] as its visual backbone. We observe that ManiSkill2 with asynchronous rendering enabled (and without the render server) is already able to outperform the speed of other frameworks. With render server enabled, ManiSkill2 further achieves 2000+ FPS with 16 parallel environments on a single GPU.

Additionally, we demonstrate the advantage of memory sharing thanks to our render server. We extend the *PickClutterYCB* environment to include 74 YCB objects per scene, and create the same setting in Habitat 2.0. As shown in Table 2.2, even though we enable GPU texture compression for Habitat and use regular textures for ManiSkill2, the memory usage of Habitat grows linearly as the number of environments increases, while ManiSkill2 requires very little memory to create additional environments since all meshes and textures are shared across environments.

2.5 Applications

In this section, we show how ManiSkill2 supports widespread applications, including sense-plan-act frameworks and learning-based approaches (reinforcement and imitation learning). In this section, for tasks that have asset variations, we report results on training objects. Results on held-out objects are presented in Sec. 2.11. Besides, we demonstrate that policies trained in ManiSkill2 have the potential to be directly deployed in the real world.

2.5.1 Sense-Plan-Act

Sense-Plan-Act (SPA) is a classical framework in robotics. Typically, a SPA method first leverages perception algorithms to build a world model from observations, then plans a sequence of actions through motion planning, and finally execute the plan. However, SPA methods are usually open-loop and limited by motion planning, since perception and planning are independently optimized. In this section, we experiment with two perception algorithms, Contact-GraspNet [108] and Transporter Networks [135].

Contact-GraspNet for *PickSingleYCB*

The SPA solution for *PickSingleYCB* is implemented as follows. First, we use Contact-GraspNet (CGN) to predict potential grasp poses along with confidence scores given the observed partial point cloud. We use the released CGN model pre-trained on ACRONYM [26]. Next, we start with the predicted grasp pose with the highest score, and try to generate a plan to achieve it by motion planning. If no plan is found, the grasp pose with the next highest score is attempted until a valid plan is found. Then, the agent executes the plan to reach the desired grasp pose and closes its grippers to grasp the object. Finally, another plan is generated and executed to move the gripper to the goal position.

For each of the 74 objects, we conduct 5 trials (different initial states). The task succeeds if the object's center of mass is within 2.5cm of the goal. The success rate is 43.24%. There are two main failure modes: 1) predicted poses are of low confidence (27.03% of trials), especially

for objects (e.g.,, spoon and fork) that are small and thin; 2) predicted poses are of low grasp quality or unreachable (29.73% of trials), but with high confidence. See Sec. 2.11.1 for examples.

Transporter Network for *AssemblingKits*

We benchmark Transporter Networks (TPN) on our *AssemblingKits*. The original success metric (pose accuracy) used in TPN is whether the peg is placed within 1cm and 15 degrees of the goal pose. Note that our version requires pieces to actually fit into holes, and thus our success metric is much stricter. We train TPN from scratch with image data sampled from training configurations using two cameras, a base camera and a hand camera. To address our high-precision success criterion, we increase the number of bins for rotation prediction from 36 in the original work to 144. During evaluation, we employ motion planning to move the gripper to the predicted pick position, grasp the peg, then generate another plan to move the peg to the predicted goal pose and drop it into the hole. The success rate over 100 trials is 18% following our success metric, and 99% following the pose accuracy metric of [135]. See Sec. 2.11.2 for more details.

2.5.2 Imitation & Reinforcement Learning with Demonstrations

For the following experiments, unless specified otherwise, we use the *delta end-effector pose* controller for rigid-body environments and the *delta joint position* controller for soft-body environments, and we translate demonstrations accordingly using the approach in Sec.2.2.2. Visual observations are captured from a base camera and a hand camera. For RGBD-based agents, we use IMPALA [27] as the visual backbone. For point cloud-based agents, we use PointNet [91] as the visual backbone. In addition, we transform input point clouds into the end-effector frame, and for environments where goal positions are given (PickCube and PickSingleYCB), we randomly sample 50 green points around the goal position to serve as visual cues and concatenate them to the input point cloud. We run 5 trials for each experiment and report the mean and standard deviation of success rates. Further details are presented in Sec. 2.11.3.

Table 2.3. Mean and standard deviation of the success rate of behavior cloning on rigid-body and soft-body tasks. For rigid-body assembly tasks not shown in the table, success rate is 0.

Obs. Mode	PickCube	StackCube	Fill	Hang	Excavate	Pour	Pinch	Write
Point Cloud	0.22 ± 0.06	0.04 ± 0.02	0.45 ± 0.04	0.35 ± 0.15	0.08 ± 0.04	0.02 ± 0.02	0.00 ± 0.00	0.00 ± 0.00
RGBD	0.01 ± 0.02	0.00 ± 0.00	0.62 ± 0.07	0.20 ± 0.12	0.21 ± 0.04	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Table 2.4. Mean and standard deviation of success rates of DAPG+PPO on rigid-body tasks. Training budget is 25M time steps.

Obs. Mode	PickCube	StackCube	PickSingleYCB	PegInsSide	PlugCharger	AssemblingKits	TurnFaucet	AvoidObstacles
Point Cloud	0.94 ± 0.03	0.95 ± 0.02	0.51 ± 0.05	0.01 ± 0.01	0.01 ± 0.02	0.00 ± 0.00	0.04 ± 0.03	0.00 ± 0.00
RGBD	0.91 ± 0.05	0.87 ± 0.04	0.18 ± 0.07	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.03 ± 0.03	0.00 ± 0.00

Imitation Learning

We benchmark imitation learning (IL) with behavior cloning (BC) on our rigid and soft-body tasks. All models are trained for 50k gradient steps with batch size 256 and evaluated on test configurations.

Results are shown in Table 2.3. For rigid-body tasks, we observe low or zero success rates. This suggests that BC is insufficient to tackle many crucial challenges from our benchmark, such as precise control in assembly and large asset variations. For soft-body tasks, we observe that it is difficult for BC agents to precisely estimate action influence on soft body properties (e.g. displacement quantity or deformation). Specifically, agents perform poorly on *Excavate* and *Pour*, as *Excavate* is successful only if a specified amount of clay is scooped up and *Pour* is successful when the final liquid level accurately matches a target line. On the other hand, for *Fill* and *Hang*, which do not have such precision requirements (for *Fill*, the task is successful as long as the amount of particles in the beaker is larger than a threshold), the success rates are much higher. In addition, we observe that BC agents cannot well utilize target shapes to guide precise soft body deformation, as they are never successful on *Pinch* or *Write*. See Sec. 2.11.5 for further analysis.

RL with Demonstrations

We benchmark demonstration-based online reinforcement learning by augmenting Proximal Policy Gradient (PPO) [100] objective with the demonstration objective from Demonstration-Augmented Policy Gradient (DAPG) [96]. Our implementation is similar to [52], and further details are presented in Sec. 2.11.3. We train all agents from scratch for 25 million time steps. Due to limited computation resources, we only report results on rigid-body environments.

Results are shown in Table 2.4. We observe that for pick-and-place tasks, point cloud-based agents perform significantly better than RGBD-based agents. Notably, on *PickSingleYCB*, the success rate is even higher than Contact-GraspNet with motion planning. This demonstrates the potential of obtaining a single agent capable of performing general pick-and-place across diverse object geometries through online training. We also further examine factors that influence point cloud-based manipulation learning in Sec. 2.11.6. However, for all other tasks, notably the assembly tasks that require high precision, the success rates are near zero. In Sec. 2.11.7, we show that if we increase the clearance of the assembly tasks and decrease their difficulty, agents can achieve much higher performance. This suggests that existing RL algorithms might have been insufficient yet to perform highly precise controls, and our benchmark poses meaningful challenges for the community.

In addition, we examine the influence of controllers for agent learning, and we perform ablation experiments on *PickSingleYCB* using point cloud-based agents. When we replace the *delta end-effector pose* controller in the above experiments with the *delta joint position* controller, the success rate falls to 0.22±0.18. The profound impact of controllers demonstrates the necessity and significance of our multi-controller conversion system.

2.5.3 Sim-to-Real

ManiSkill2 features fully-simulated dynamic interaction for rigid-body and soft-body. Thus, policies trained in ManiSkill2 have the potential to be directly deployed in the real world.

PickCube: We train a visual RL policy on *PickCube* and evaluate its transferability to the

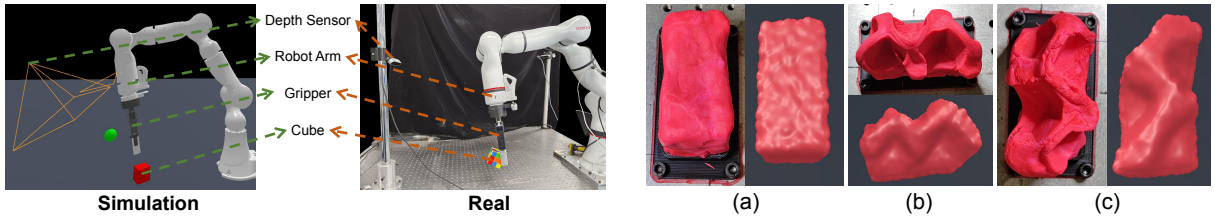


Figure 2.4. Left: Simulation and real world setup for *PickCube*. Right: Motion planning execution results for *Pinch* in simulation and in the real world: (a) initial state; (b) letter "M"; (c) letter "S".

real world. The setup is shown in Fig. 2.4-L, which consists of an Intel RealSense D415 depth sensor, a 7DoF ROKAE xMate3Pro robot arm, a Robotiq 2F-140 2-finger gripper, and the target cube. We first acquire the intrinsic parameters for the real depth sensor and its pose relative to the robot base. We then build a simulation environment aligned with the real environment. We train a point cloud-based policy for 10M time steps, where the policy input consists of robot proprioceptive information (joint position and end-effector pose) and uncolored point cloud backprojected from the depth map. The success rate in simulation is 91.0%. We finally directly evaluate this policy in the real world 50 times with different initial states, where we obtain 60.0% success rate. We conjecture that the performance drop is caused by the domain gap in depth maps, as we only used Gaussian noise and random pixel dropout as data augmentation during simulation policy training.

Pinch: We further evaluate the fidelity of our soft-body simulation by executing the same action sequence generated by motion planning in simulation and in the real world and comparing the final plasticine shapes. Results are shown in Fig. 2.4-R, which demonstrates that our 2-way coupled rigid-MPM simulation is able to reasonably simulate plasticine's deformation caused by multiple grasps.

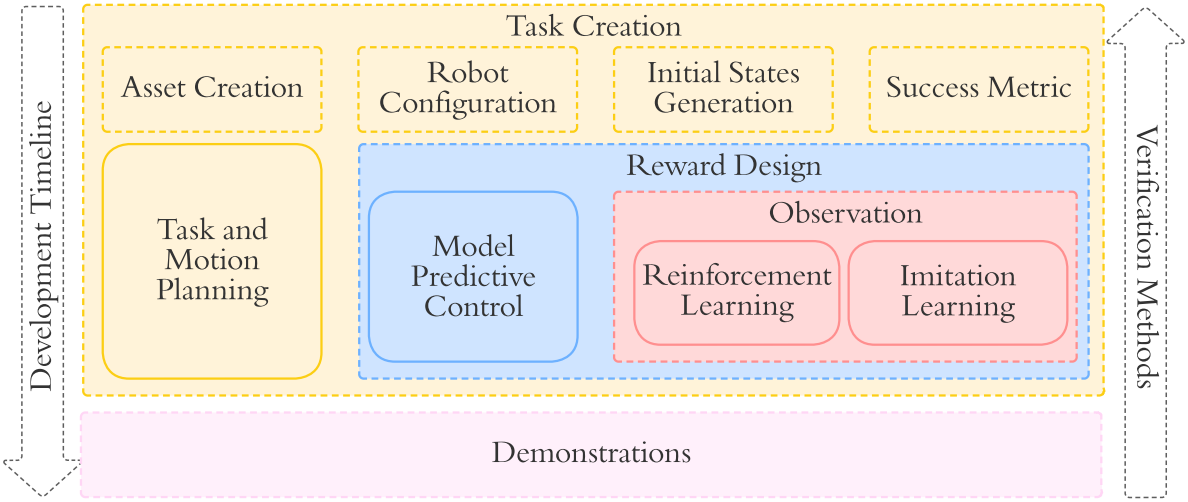


Figure 2.5. The workflow to build environments for generalizable manipulation skills.

2.6 System Design for Development and Evaluation

2.6.1 Verification-driven Iterative Development

Our workflow, following agile software development [5], highlights a verification-driven iterative development process. Different approaches can be used to generate demonstrations according to characteristics and difficulties of tasks, which verifies environments as a byproduct. The workflow is also designed to be scalable and affordable to continuous integration of assets and tasks. Fig 2.5 illustrates the workflow.

Our workflow consists of 3 stages: task creation, reward design, and observation configuration. The first stage focuses on building task essentials, including creating assets, (e.g., convex decomposition [118], texture baking), configuring robots, generating initial states, and defining success metrics. The second stage aims at prototyping shaped reward functions. The reward function is a requisite for methods like Model-Predictive Control (MPC) and RL. The third stage addresses observation spaces. For instance, camera parameters and placements need to be tailored for tasks so that observations contain adequate information. To collect demonstrations and verify environments, we employ one of or a mixture of 3 complementary approaches: task and motion planning (TAMP), MPC, and RL.

Different methods have their own advantages and disadvantages. TAMP is free of crafting reward, and is suitable for many stationary manipulation tasks like pick-and-place, but shows difficulty when tackling underactuated systems (e.g. pushing chairs and moving buckets in [82]). MPC is able to search solutions to difficult tasks given well-designed shaped rewards without training or observations. However, it is non-trivial to design a universal shaped reward for a variety of objects. RL requires additional training and hyperparameter tuning, but is more scalable than MPC during inference.

2.6.2 Cloud Based Evaluation System

To allow the community to evaluate and benchmark together we build and provide a simple cloud based evaluation system. Users can register accounts and enter the benchmark and begin making submissions that solve our various tasks.

A key feature of the evaluation system is flexibility. This is increasingly important as the number of unique approaches from heuristics, motion planners, and end-to-end RL solutions grow. Evaluation systems need the flexibility to allow all kinds of approaches to be benchmarked. To this end, we do the following.

User submissions are in the form of docker images, allowing users to install any code and save any models necessary to solve various tasks. This makes programming on the user's side very flexible. The user only needs to provide a function that accepts observations and returns actions.

We further give users the flexibility to configure the evaluation environment to suit their needs before running the evaluation. For example, users can define a configuration function in their solution that sets the observation mode (e.g. RGB-D or Point Cloud), as well as controllers (e.g. *delta end-effector pose* or *joint position*).

To benchmark a submitted docker image, we simply pull it to our server and run the evaluation code that loads the user's solution. The results of the evaluation are then submitted to a database and displayed on a pubic benchmark.

2.7 Details of the Comprehensive Controller Suite

Controllers are interfaces between policies and robots. Policies output actions to controllers, and controllers convert actions to control signals to drive the robot joints. In ManiSkill2, the default robot being controlled is the Franka Emika, also known as Panda. The degree of freedom (DoF) of a single Panda arm is 7.

2.7.1 Terminology

1. **Fixed joint**: A joint that cannot be controlled. The degree of freedom (DoF) is 0.
2. **qpos** (q): Controllable joint positions.
3. **qvel** ($\dot{q}$): Controllable joint velocities.
4. **Target joint positions** ($\bar{q}$): Target positions of motors that drive each joint.
5. **Target joint velocities** ($\bar{\dot{q}}$): Target velocities of motors that drive each joint.
6. **End-effector position** (p): The position of an end-effector.
7. **End-effector rotation** (R): The rotation of an end-effector.
8. **End-effector target position** ($\bar{p}$): The target position of an end-effector.
9. **End-effector target rotation** ($\bar{R}$): The target rotation of an end-effector.
10. **PD controller**: Control loop based on $\tau(t) = K_p(\bar{q}(t) - q(t)) + K_d(\bar{\dot{q}}(t) - \dot{q}(t))$. K_p (stiffness) and K_d (damping) are hyperparameters. $\tau(t)$ denotes the torque (generalized force) of the motors.
11. **Augmented PD controller**: Augmented PD controller: Passive forces (like gravity) are compensated for the PD controller.
12. **Action** (a): Input to the controllers, and also output of the policy.
13. **Tool center point** (TCP): TCP is a user defined coordinate frame, often relatively fixed to the robot end effector. For example, in our case, if the robot uses a two-finger gripper, TCP

is defined at the center point between the gripper's two fingers.

2.7.2 Target vs. Non-Target Controllers

In our controller implementation, we have the notion of "target" and "non-target" controllers. For example, when we say *delta end-effector pose* controller, the new desired pose is specified relative to the current end-effector pose. In contrast, if we say *target delta end-effector pose* controller, the new desired pose is specified relative to the previous desired pose. Please read the next section for their formal definitions.

2.7.3 Normalized Action Space

We design the action space of controllers to conform to the preferences of users. As RL users generally prefer normalized action space, most controllers listed below will have a normalized action space $[-1, 1]$, with a few exceptions listed individually.

2.7.4 Details of Controllers

Arm Controllers

1. *arm_pd_joint_pos* (7-dim, unnormalized): $a_t = \bar{q}_t$. The target joint velocities $\dot{q}_t$ are always 0. As this controller is suitable for motion planning, the action space of this controller is not normalized.
2. *arm_pd_joint_delta_pos* (7-dim): $a_t = \bar{q}_t - q_{t-1}$.
3. *arm_pd_joint_target_delta_pos* (7-dim): $a_t = \bar{q}_t - \bar{q}_{t-1}$.
4. *arm_pd_ee_delta_pos* (3-dim): $a_t = \bar{p}_t - p_{t-1}$, where p_t is the position of the end-effector at timestep t. The controller then internally computes the target joint positions of the robot: $q_t = IK(\bar{p}_t, \bar{R}_t)$, where $IK(\cdot, \cdot)$ computes the joint positions from the end-effector's position and rotation before sending the joint positions to the PD controller. Note that this controller only controls the position, but not the rotation, of the end-effector.
5. *arm_pd_ee_target_delta_pos* (3-dim): $a_t = \bar{p}_t - \bar{p}_{t-1}$

6. *arm_pd_ee_delta_pose* (6-dim): This controller is very similar to the previous *arm_pd_ee_delta_pos* controller with the addition of end-effector's rotation being passed in as input. Thus $\bar{T}_t = T_a \cdot T_{t-1}$, where $\bar{T}$ is the target transformation of the end-effector, and T_a is the delta pose induced by the 3D position and 3D rotation (represented as compact axis-angle) of the action.
7. *arm_pd_ee_target_delta_pose* (6-dim): $\bar{T}_t = T_a \cdot \bar{T}_{t-1}$.
8. *arm_pd_joint_vel* (7-dim): $a_t = \bar{q}_t$. The stiffness value K_p of this controller is always set to 0.
9. *arm_pd_joint_pos_vel* (14-dim): An extension of *arm_pd_joint_pos* that supports target velocities input.
10. *arm_pd_joint_delta_pos_vel* (14-dim): The delta variant of the *arm_pd_joint_pos_vel* controller.

Gripper Controller (1-dim)

We use joint position control for the gripper, and we force the two gripper fingers to have the same target position.

2.7.5 Effectiveness of Conversion of Demonstration Action Spaces

In this section, we exemplify the success rate of our demonstration conversion method by converting from the *arm_pd_joint_pos* controller to the *arm_pd_ee_delta_pose* controller. A demonstration is converted successfully if, following the same trajectory initialization and the converted actions, the task is successful at the last time step. We experiment on 4 representative tasks: *PickSingleYCB*, *AssemblingKits*, *TurnFaucet*, *Write*. For each task, we select 100 demonstrations randomly. The success rates for *PickSingleYCB*, *AssemblingKits*, *Write*, *TurnFaucet* are 99%, 98%, 100%, and 80%, respectively. Note that our policy to generate demonstrations for *TurnFaucet* involves rich and inconsistent contact between the end-effector and the faucet handle (i.e., our policy uses the gripper to push the handle, rather than grasping

and rotating it, in which case force closure can lead to more consistent contact). Such polices can be sensitive to accumulated errors during execution, which can result in task failure, although the actions converted by our demonstration conversion method attempt to reproduce motion faithfully.

2.8 Details of Observations, Task Families, Demonstrations and Evaluation Protocols

Unless otherwise noted, all demonstrations are generated through TAMP. **For evaluation, we employ a two-stage setup. Final result is the average result from the two stages.**

2.8.1 Supported Observation Modes

We support most observation modes found in OpenAI gym [10]. The details of each observation mode are listed below.

1. *state_dict*: Returns a dictionary of states that contains robot proprioceptive information, ground truth object information (such as object poses), and task-specific goal information (if given). Visual observations (images and point clouds) are **not** included.

2. *state*: Returns the flattened version of a *state_dict*.

3. *rgbd*: Returns rendered RGBD images from all cameras, along with robot proprioceptive information and task-specific goal information (if given).

4. *rgbd_robot_seg*: On top of *rgbd*, returns ground truth segmentation masks for the robot joints.

5. *pointcloud*: Returns a fused point cloud from all cameras, along with robot proprioceptive information and task-specific goal information (if given).

6. *pointcloud_robot_seg*: On top of *pointcloud*, returns ground truth segmentation masks for the robot joints.

Here, the robot proprioceptive information includes joint positions, joint velocities, the pose of the robot base, along with the pose of the gripper's tool center point if the robot uses a two-finger gripper.

Note that different from the previous version of ManiSkill, ManiSkill2 does not include ground-truth segmentation in the default observation modes (*rgbd* or *pointcloud*). All visual-based experiments in this paper do not leverage such privileged information. For example, to specify which link of a faucet should be manipulated in *TurnFaucet*, we use its initial position instead of a ground-truth segmentation mask. Besides, we also support observations modes (*rgbd+robot_seg*, *pointcloud+robot_seg*) to provide the segmentation masks of robot links, which facilitates robotic applications and can be obtained in the real world using the robot proprioceptive information.

2.8.2 Pick-and-Place

PickCube

- Objective: Pick up a cube and move it to a goal position.
- Success Metric: The cube is within 2.5 cm of the goal position, and the robot is static.
- Demonstration Format: 1,000 successful trajectories.
- Evaluation Protocol: 100 episodes with different initial joint positions of the robot and initial cube pose for each of stage 1 and stage 2.
- Task-specific Extra Observations: 3D goal position of the cube.

StackCube

- Objective: Pick up a red cube and place it onto a green one.
- Success Metric: The red cube is placed on top of the green one stably and it is not grasped.
- Demonstration Format: 1,000 successful trajectories.
- Evaluation Protocol: 100 episodes with different initial joint positions of the robot and initial poses of both cubes for each of stage 1 and stage 2.

- Task-specific Extra Observations: None.

PickSingleYCB

- Objective: Pick up a YCB object and move it to a goal position.
- Success Metric: The object is within 2.5 cm of the goal position, and the robot is static.
- Demonstration Format: 100 successful trajectories for each of the 74 YCB objects.
- Evaluation Protocol: In addition to the training objects, we also use another confidential set of 40 objects from other sources as the test object set. For the two evaluation stages in total, for each object in the training set, we test 5 episodes with different seeds. For each object in the test set, we test 10 episodes with different seeds. Half of the objects are evaluated in each stage.
- Task-specific Extra Observations: 3D goal position of the object.

PickSingleEGAD

- Objective: Pick up an EGAD object and move it to a goal position. The color for the EGAD object is randomized.
- Success Metric: The object is within 2.5 cm of the goal position, and the robot is static.
- Demonstration Format: 5 trajectories for each of the 1,600 training objects sampled from EGAD. For certain objects where it's difficult to apply TAMP, we might provide less than 5 trajectories.
- Evaluation Protocol: For this task, we have held out a portion of the EGAD dataset. This held-out test dataset consists of 150 objects. During evaluation, in each stage, we evaluate 1 trajectory for each of the 150 objects sampled from the training dataset and 2 trajectories for each of the 75 objects sampled from the held-out test dataset.
- Task-specific Extra Observations: 3D goal position of the object.

PickClutterYCB

- Objective: Pick up an object from a clutter of 4-8 YCB objects.
- Success Metric: The object is within 2.5 cm of the goal position, and the robot is static.
- Demonstration Format: A total of 4986 trajectories from the training object set.
- Evaluation Protocol: In addition to the training objects, we also use another confidential set of 40 objects from other sources as the test object set. For each evaluation stage, we test 100 episode configurations on the training object set and on the test object set.
- Task-specific Extra Observations: 3D position of the object to pick up, and 3D position of the goal.

2.8.3 Assembly

AssemblingKits

- Objective: Insert an object into the corresponding slot on a plate.
- Success Metric: An object must fully fit into its slot, which must simultaneously satisfy 3 criteria: (1) height of the object center is within 3mm of the height of the plate; (2) rotation error is within 4 degrees; (3) position error is within 2cm.
- Demonstration Format: We provide 1,720 trajectories in total. These trajectories are generated from over 300 kit configurations and 20 training shapes.
- Evaluation Protocol: This task has a held-out test dataset for evaluation. The test dataset features 20 shapes that are similar to the shapes in the training set. We provide samples for test assets in Fig. 2.6. In each evaluation stage, we evaluate on 100 sampled training episode configurations and 100 sampled test dataset configurations.
- Task-specific Extra Observations: 3D initial and goal position of the object to be placed.

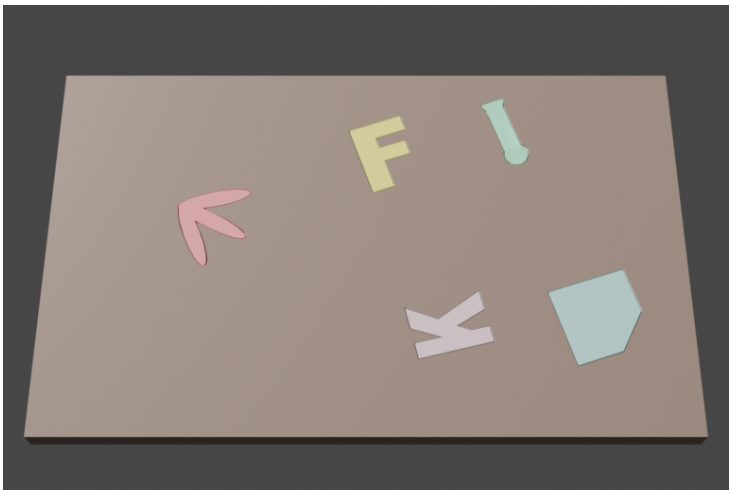


Figure 2.6. A sample plate with test assets for *AssemblingKits*.

PegInsertionSide

- Objective: Insert a peg into the horizontal hole in a box.
- Success Metric: Half of the peg is inserted into the hole.
- Demonstration Format: 1,000 successful trajectories.
- Evaluation Protocol: 100 episodes with different initial joint positions of the robot, initial poses of the peg and box, the position and size of the hole for each of stage 1 and stage 2.
- Task-specific Extra Observations: None.

PlugCharger

- Objective: Plug a charger into a wall receptacle.
- Success Metric: The charger is fully inserted into the receptacle.
- Demonstration Format: 1,000 successful trajectories.
- Evaluation Protocol: 100 episodes with different initial joint positions of the robot, initial poses of the charger and wall for each of stage 1 and stage 2.
- Task-specific Extra Observations: None.

2.8.4 Miscellaneous Tasks

AvoidObstacles

- Objective: Navigate the robot arm through a region of dense obstacles and move the end-effector to a goal pose. The shape and color of dense obstacles are randomized.
- Success Metric: The end-effector pose is within 2.5 cm and 15 degrees of the goal pose.
- Demonstration Format: 1976 trajectories for different layouts.
- Evaluation Protocol: 100 episodes with different layouts of obstacles for each of stage 1 and stage 2.
- Task-specific Extra Observations: The goal pose of the end-effector.

TurnFaucet

- Objective: Turn on a faucet by rotating its handle.
- Success Metric: The faucet handle has been turned past a target angular distance.
- Demonstration Format: For most faucet models, we provide 100 trajectories per asset. For approximately 15 of the 60 training models were TAMP cannot find a solution, demonstrations are generated through MPC-CEM using our designed rewards.
- Evaluation Protocol: This task has a held-out test object set. For the two evaluation stages in total, we evaluate 5 episodes for each of the 60 training objects and 17 episodes for each of the 18 test objects. Half of the objects are evaluated in each stage.
- Task-specific Extra Observations: The remaining angular distance to rotate the handle, the target handle position (since there can be multiple handles in a single faucet), and the direction to rotate the handle specified as 3D joint axis.

2.8.5 Soft-body Manipulation

Fill

- Objective: Fill clay from a bucket into the target beaker.

- Success Metric: The amount of clay inside the target beaker > 90%; soft body velocity < 0.05.
- Demonstration Format: 200 successful trajectories generated through motion planning.
- Evaluation Protocol: 100 episodes with different initial rotations of the bucket and initial positions of the beaker for each of stage 1 and stage 2.
- Task-specific Extra Observations: Beaker position.

Hang

- Objective: Hang a noodle on a target rod.
- Success Metric: Part of the noodle is higher than the rod; two ends of the noodle are on different sides of the rod; the noodle is not touching the ground; the gripper is open; soft body velocity < 0.05.
- Demonstration Format: 200 successful trajectories generated through motion planning.
- Evaluation Protocol: 100 episodes with different initial positions of the gripper and rod poses for each of stage 1 and stage 2.
- Task-specific Extra Observations: Rod position.

Excavate

- Objective: Lift a specific amount of clay to a target height.
- Success Metric: The amount of lifted clay must be within a given range; the lifted clay is higher than a specific height; fewer than 20 clay particles are spilled on the ground; soft body velocity < 0.05.
- Demonstration Format: 200 successful trajectories generated through motion planning.
- Evaluation Protocol: 100 episodes with different bucket poses and initial heightmaps of clay for each of stage 1 and stage 2.
- Task-specific Extra Observations: Target clay amount.

Pour

- Objective: Pour liquid from a bottle into a beaker.
- Success Metric: The liquid level in the beaker is within 4mm of the red line; the spilled water is fewer than 100 particles; the bottle returns to the upright position in the end; robot arm velocity < 0.05.
- Demonstration Format: 200 successful trajectories generated through motion planning.
- Evaluation Protocol: 100 episodes with different bottle positions, the level of water in the bottle, and beaker positions for each of stage 1 and stage 2.
- Task-specific Extra Observations: Red line height.

Pinch

- Objective: Deform plasticine into a target shape.
- Success Metric: The Chamfer distance between the current plasticine and the target shape is less than $0.3t$, where t is the Chamfer distance between the initial shape and target shape.
- Demonstration Format: 1556 successful trajectories generated through heuristic motion planning. Different trajectories correspond to different target shapes.
- Evaluation Protocol: 50 episodes with different target shapes for each of stage 1 and stage 2.
- Task-specific Extra Observations: RGBD / point cloud observations of the target plasticine from 4 different views.

Write

- Objective: Write a given character on clay. The target character is randomly sampled from an alphabet of over 50 characters.
- Success Metric: The IoU (Intersection over Union) between the current pattern and the target character is larger than 0.8.
- Demonstration Format: 200 successful trajectories generated through heuristic motion planning.

- Evaluation Protocol: 50 episodes with different target characters for each of stage 1 and stage 2.
- Task-specific Extra Observations: The depth map of the target character.

2.8.6 Mobile Manipulation

OpenCabinetDrawer

- Objective: A single-arm mobile robot needs to open a designated target drawer on a cabinet. The friction and damping parameters for the drawer joints are randomized.
- Success Metric: The target drawer has been opened to at least 90% of the maximum range, and the target drawer is static.
- Demonstration Format: 300 trajectories for each target drawer in the training object set. The training object set consists of 25 cabinets. Each cabinet could contain multiple drawers.
- Evaluation Protocol: This task has a held-out test object set (10 unseen cabinets). In the first stage, we evaluate 250 trajectories in total. Among these 250 trajectories, 125 levels are evenly distributed over 5 unseen objects in the test set, and the other 125 levels are evenly distributed over all objects in the training set. In the second stage, we evaluate another 250 trajectories. Similarly, 125 levels come from the training set and the other 125 levels from the 5 other unseen objects in the test set (different from the 5 test objects in stage 1).
- Task-specific Extra Observations: Since one cabinet can contain several drawers, we specify the target drawer by its initial center of mass.

OpenCabinetDoor

- Objective: A single-arm mobile robot needs to open a designated target door on a cabinet. The friction and damping parameters for the door joints are randomized.
- Success Metric: The target door has been opened to at least 90% of the maximum range, and the target door is static.

- Demonstration Format: 300 trajectories for each target door in the training object set. The training object set consists of 42 cabinets. Each cabinet could contain multiple doors.
- Evaluation Protocol: This task has a held-out test object set (10 unseen cabinets). The evaluation protocol is similar to *OpenCabinetDrawer*.
- Task-specific Extra Observations: Since one cabinet can contain several doors, we specify the target door by a segmentation mask.

PushChair

- Objective: A dual-arm mobile robot needs to push a swivel chair to a target location on the ground (indicated by a red hemisphere) and prevent it from falling over. The friction and damping parameters for the chair joints are randomized.
- Success Metric: The chair is close enough (within 15 cm) to the target location, is static, and does not fall over.
- Demonstration Format: 300 trajectories for each chair in the training object set. The training object set consists of 26 chairs.
- Evaluation Protocol: This task has a held-out test object set (10 unseen chairs). The evaluation protocol is similar to *OpenCabinetDrawer*.
- Task-specific Extra Observations: None.

MoveBucket

- Objective: A dual-arm mobile robot needs to move a bucket with a ball inside and lift it onto a platform.
- Success Metric: The bucket is placed on or above the platform at the upright position, and the bucket is static, and the ball remains in the bucket.
- Demonstration Format: 300 trajectories for each bucket in the training object set. The training object set consists of 29 buckets.

- Evaluation Protocol: This task has a held-out test object set (10 unseen buckets). The evaluation protocol is similar to *OpenCabinetDrawer*.
- Task-specific Extra Observations: None.

2.9 Details of Soft-Body Simulation and Rendering

2.9.1 Soft-Body Simulation and 2-Way Coupling Algorithm

The simulation and 2-way coupling algorithm are summarized in algorithm 1.

Algorithm 1. Rigid MPM Simulation and Dynamic Coupling

```
initialize rigid scene
initialize soft scene
copy rigid body shapes and center of mass to soft scene
initialize renderer
for environment step do
    execute ManiSkill2 controllers
    for rigid step per environment step do
        process ManiSkill2 substep
        step rigid scene
        copy rigid body poses to soft scene
        initialize force-torque buffers per rigid body
        for soft step per rigid step do
            compute penalty forces
            accumulate equivalent rigid-body forces and torques in force-torque buffers
            MPM particle to grid (mass, momentum, forces)
            MPM grid compute velocity
            MPM grid to particle (velocity)
            integrate MPM particles
        end for
        apply accumulated forces and torques on rigid bodies
    end for
    copy rigid body states to SAPIEN renderer
    copy MPM particles to SAPIEN renderer
    execute renderer
end for
```

2.9.2 Soft-Body Rendering

To support visual learning, we extended SAPIEN's renderer to support rendering particle-based soft body. To render particle-based material, one common approach is to convert the particles to triangle meshes using a meshing algorithm such as marching cubes; PlasticineLab implements a ray-tracing framework that renders spheres directly. Our approach is screen-space splatting [20], similar to Nvidia Flex's built-in renderer. We customize SAPIEN's shader to render soft-body particles as spheres, use a bilateral filter to smooth the depth buffer, then compute normal and lighting on the smoothed soft-body depth. These are implemented as extra screen-space render passes. The effect of the smoothing filter is shown in figure 2.7. Moreover, we customize Warp to support the transfer of particle positions from simulation to renderer with a single GPU-GPU copy; this further reduces rendering latency. A concern of the screen-space splatting is the inconsistency across different views due to the use of screen-space filters. However, in practice, by scaling the bilateral filter according to pixel distance from camera, the rendering results produced are visually consistent most of the time.

(a) Without bilateral filter **(b)** With bilateral filter

Figure 2.7. Comparison between results of our particle renderer without and with bilateral filter.

2.9.3 Other Implementation Details

We summarize the key parameters In table 2.5. For all soft body environments with rendering, a single environment runs at around 17-18 FPS; 16 parallel environments on a single GPU gives overall 80-84 FPS on a single RTX Titan GPU (4x real time). This performance

gain is mainly due to the sequential CPU-rigid-body and GPU-soft-body simulation. It is also partially due to a single CPU processor not able to submit CUDA kernels fast enough to keep up with the GPU. Therefore, it can potentially be further optimized with vectorized simulation, which we leave as future work.

Table 2.5. Ranges for key parameters used in our MPM simulation. All fields are in SI units.

Grid Length	Particle Volume	Density	Young's Modulus	Poisson Ratio	Yield Stress
0.005 to 0.015	6.2e-8/1.2e-7	300 to 3000	1e4/3e5	0.3	2e3/1e4

2.10 Details of Performance Optimization

2.10.1 Render Server Implementation

Our render server is implemented with the gRPC framework, which exchanges Protocal Buffers with the HTTP 2 protocol over networks or unix sockets. The server side is managed by a thread pool, listening to client requests on multiple concurrent threads. For a "take picture" request from a worker process, our implementation puts the task of updating GPU matrices and launching draw calls onto another thread and returns immediately back to the worker process. This ensures minimum waiting time on the worker side.

On the rendering server, all rendering resources are managed by a central resource manager. Any resource loading request (e.g., models, images, textures) must go through the manager. The manager ensures only one copy of any resource is loaded onto the GPU, shared across potentially multiple scenes.

2.10.2 Additional Benefits of Render Server

In general, rendering resource sharing across processes is not well supported. Rendering frameworks like OpenGL and Vulkan are designed to be efficient in a single-process multi-threaded environment. Resources and documentations on multi-process renderer are rarely

provided. Therefore as a developer, it is far easier to understand renderers in a single process setting.

In addition, Nvidia's GPU profiler, Nsight System, is currently unable to profile Vulkan in multiple processes on Linux in our experiments. Running the renderer in multiple processes makes it hard to understand GPU performance. Thus, running rendering in one main process is almost required for a developer to optimize for hardware utilization.

2.10.3 More Details on Sample Collection Speed Comparison

We use the *PickCube* environment to compare the sample collection speed between different simulators. The resulting total number of vertices of collision meshes is 1009 for ManiSkill2 and Habitat-2.0 and 1210 for IsaacGym and Robosuite. The number of vertices of visual meshes is 69747 for all simulators. Each control step consists of 25 physical simulation steps. We use the GPU pipeline for rendering. All frameworks are given a budget of 16 CPU cores(logical processors) of an Intel i9-9960X CPU with 128G memory and 1 RTX Titan GPU with 24G memory.

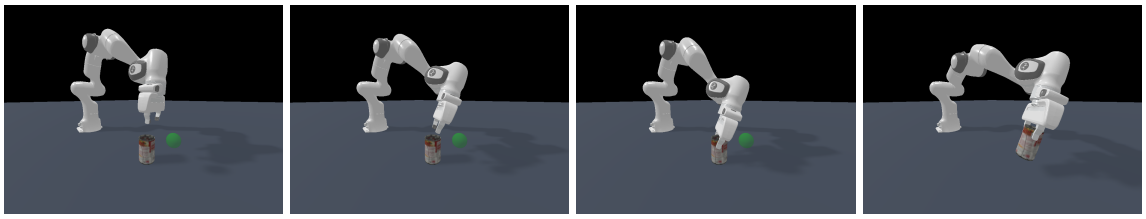
2.11 Additional Experiment Details, Results, and Analysis

2.11.1 Contact-GraspNet for *PickSingleYCB*

We directly use the pretrained model provided by the original authors of Contact-GraspNet [108]. We exemplify successful trajectories by the model in Fig. 2.8, as well as common failure modes in Fig. 2.9.

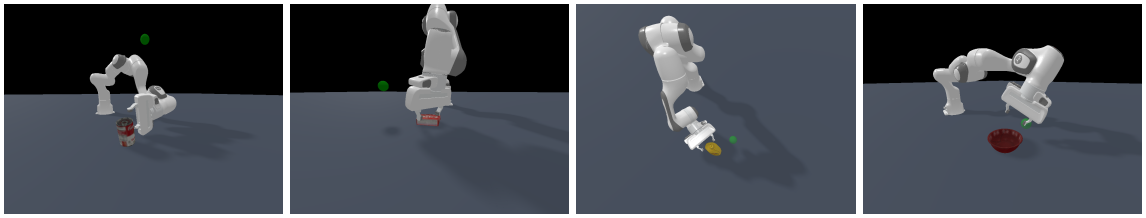
2.11.2 Transporter Network for *AssemblingKits*

We adopt the official code from [135]. To encourage precise rotation prediction, we increase the number of rotation prediction bins from 36 in the original work to 144. We further benchmark with two grippers: a suction gripper following the original work, and another two-finger gripper from Franka Panda. Our training dataset is generated from random initial



(a) **(b)** **(c)** **(d)**

Figure 2.8. Sampled frames demonstrating a correct and successful grasp of a can. Frame (a) shows the initial state; (b) shows the gripper approaching the predicted grasp from above; (c) shows the gripper grasping the can; (d) shows the robot moving the can towards the green goal position.



(a) **(b)** **(c)** **(d)**

Figure 2.9. Examples of unsuccessful grasps. (a) shows an erroneous rotation prediction in grasp pose; (b) shows a correct rotation prediction in grasp pose, but the gripper is not close enough to the object to grasp it; (c) shows a reasonable grasp pose, but the gripper will slip away from the bottle upon finger closure due to friction and bottle geometry; (d) shows a reasonable grasp pose that is not achievable through motion planning due to kinematic constraints of the robot arm.

configurations of training assets in AssemblingKits. More specifically, for each sampled initial configuration, we capture RGBD images from the hand-camera and the base-camera, unproject them to 3D point clouds, fuse the point clouds, and render the top-down orthographic image to feed to the Transporter Network model. The ground truth labels, namely the object pick position and the goal pose to place the object, are directly obtained from the environment state information.

Table 2.6 shows the results. The success rate of Transporter Network following our success criterion (which requires pieces to fully fit in holes) is a lot lower than following the metric in the original paper. We observe that failure modes are mainly due to imprecise rotation/position prediction for placing the target piece.

Table 2.6. Success rate of Transporter Networks on our AssemblingKits task on training assets (up) and test assets (down). We also report the success rate based on the original paper, where a trial is successful if the target piece is placed within 1cm and 15 degrees of the goal position.

Gripper Type	Success (ours)	Success (Zeng et al.)	Pos < 5mm	Pos < 2.5mm	Pos < 1mm	Rot < 4°	Rot < 1°	Rot < 0.25°
Two-finger Gripper	0.18	0.98	0.98	0.80	0.30	0.96	0.48	0.22
Suction Gripper	0.15	0.93	0.89	0.66	0.18	0.91	0.48	0.22

Gripper Type	Success (ours)	Success (Zeng et al.)	Pos < 5mm	Pos < 2.5mm	Pos < 1mm	Rot < 4°	Rot < 1°	Rot < 0.25°
Two-finger Gripper	0.18	0.99	0.99	0.82	0.31	0.96	0.48	0.24
Suction Gripper	0.14	0.96	0.92	0.66	0.17	0.94	0.52	0.31

2.11.3 Detailed Setup for Imitation Learning & RL from Demonstrations

In ManiSkill2, our demonstrations are provided using the *joint position* controller. Before we train demonstration-based agents on rigid & soft-body tasks, we first use the approach in Sec.2.2.2 to translate the provided demonstrations into the *delta end-effector pose* controller for rigid-body environments, and into the *delta joint position* controller for soft-body environments. We then filter the translated trajectories, such that only successful trajectories are used for agent learning.

For RGBD-based agents, we use IMPALA [27] as the visual backbone, and we concatenate images captured from the base camera and the hand camera as visual input. Image resolution is 128x128. For point cloud-based agents, we use PointNet [91] as the visual backbone. We first obtain a single fused point cloud by transforming point clouds from different cameras into the robot-base frame and concatenating the points together. We then remove the ground using height clip and randomly downsample the point cloud to 1200 points. For rigid-body environments, we also transform the point cloud (along with robot proprioceptive information and goal position, if given) into the end-effector frame. In addition, for environments where goal positions are given (PickCube and PickSingleYCB), we randomly sample 50 green points around the goal position to serve as visual cues and concatenate them to the input point cloud.

To train demonstration-based agents, for rigid-body environments, we use 1000 demonstration trajectories, except environments that have multiple assets (PickSingleYCB: 7300 trajectories;

Table 2.7. Mean and standard deviation of success rates of DAPG+PPO on rigid-body tasks on held-out test objects. Training budget is 25M time steps.

Obs. Mode	PickSingleYCB	AssemblingKits	TurnFaucet	AvoidObstacles
Point Cloud	0.36 ± 0.06	0.00 ± 0.00	0.01 ± 0.01	0.00 ± 0.00
RGBD	0.08 ± 0.03	0.00 ± 0.00	0.01 ± 0.01	0.00 ± 0.00

TurnFaucet: 4500 trajectories; AssemblingKits: 1700 trajectories). For soft-body environments, we use 200 demonstration trajectories (except *Pinch* with 1550 trajectories).

In this work, our demonstration-based online RL algorithm is modified from Proximal Policy Gradient (PPO) [100] and Demonstration-Augmented Policy Gradient (DAPG) [96]. We adopt the training objective modified from [52]. Here, we use ρ_D and ρ_π to denote the distribution of demonstration transitions and online environment rollout transitions, respectively. We can then write the overall policy loss as follows (value loss omitted here):

$$\mathcal{L}_\rho^{CLIP}(\theta) = -\mathbb{E}_{(s,a)\sim\rho}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}(s,a), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(s,a)\right)\right]$$

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

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

We set $\omega = 0.1 \cdot 0.995^N$, where N is the epoch count for PPO. A PPO epoch is defined as online environment sampling steps followed by policy and value network updates.

For further details about networks and algorithm hyperparameters, see Sec. 2.11.8.

2.11.4 Results for DAPG+PPO on Held-Out Object Sets

In Section 2.5.2, for tasks that have asset variations, we presented evaluation results on the training object set. In this section, we present results on the held-out object set. See Table 2.7 for details.

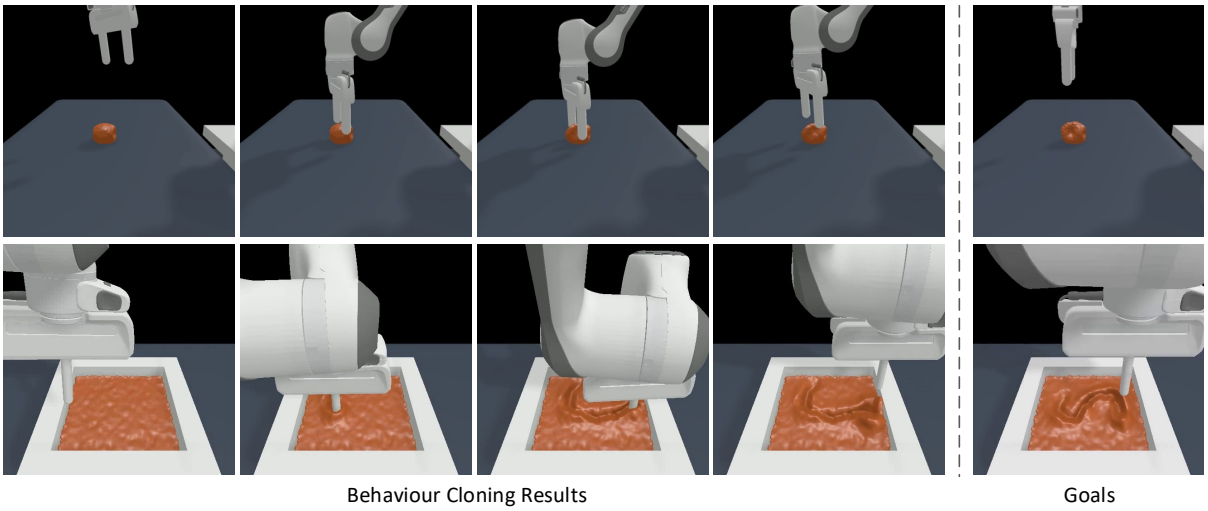


Figure 2.10. Behaviour cloning examples for *Pinch* and *Write* tasks. BC models have learned to make some progress towards the goals but not enough to meet the success conditions.

2.11.5 Further Analysis of Imitation Learning on Soft-Body Tasks

We observe that it is difficult for BC agents to accurately estimate the influence of an action on fine-grained soft body properties, such as displacement quantity and deformation. For example, both *Fill* and *Pour* tasks require a robot agent to move soft body objects (clay or liquid) into a target container, but *Fill* has a much higher success rate. An underlying cause is that *Fill* allows the robot agent to put all of the clay into the beaker while *Pour* requires higher precision i.e. the final liquid level must match the target line. Therefore, agents need to precisely control the bottle tilt angle in order to precisely control the amount of liquid poured into the beaker. Similarly, for *Excavate*, agents must perform fine judgments on how deep they should dig in order to scoop up a specified amount of clay. On the other hand, *Hang* does not require an agent to perform high accuracy measurements, so it is easier for agents to succeed.

In addition, we notice that BC agents cannot well utilize target shapes to guide precise soft body deformation. Specifically, for *Pinch* and *Write* that require shape deformation, the BC models have very poor performance. As shown in Fig. 2.10, the robot learns the motion of pinching and has made some progress toward the goal, but it is not good enough. Similarly, the robot agent has learned to draw some patterns, but they are not close enough to the target

Table 2.8. Ablations on PickSingleYCB (training object set) for point cloud-based agents trained with DAPG+PPO. The "original" result refers to the result in Table 2.4, which uses the *delta end-effector pose* controller, transforms inputs point clouds into the robot's end-effector frame, and (for pick-and-place tasks where goal position is given) appends 50 green points sampled around the goal position to the input point clouds to serve as visual cues.

Original	*Delta Joint Position* Controller	Robot Base Frame Point Cloud	Remove Visual Goal Cues
0.51 ± 0.05	0.22 ± 0.18	0.00 ± 0.00	0.16 ± 0.07

character.

2.11.6 More Results on Point Cloud-Based Manipulation Learning

In this section, we examine factors that influence point cloud-based manipulation learning to complement the results in Section 2.5.2. Results are shown in Table 2.8. In addition to our prior observation that choices of controllers have a significant effect on performance, we also observe that (1) selecting good coordinate frames to represent input point clouds could be crucial for agents' success, which corroborates the findings in [66]; (2) adding proper visual cues could benefit point cloud-based agent learning.

2.11.7 More Analysis on Assembly Tasks

In Section 2.5.2, we observed that agents trained with Imitation Learning or Reinforcement Learning perform poorly on many tasks that require high precision, such as the assembly tasks. We further analyze sources of difficulty from these tasks by training agents on easier versions of these tasks where the clearance threshold is significantly increased. Results are shown in Table 2.9. We observe that when we increase clearance threshold and decrease task difficulty, some agents are able to achieve a lot higher performance. However, even in this case, many agents still do not perform well. We conjecture that this is due to many other challenging factors, such as occlusions of the target slots when agents attempt to insert a peg or a charger.

Table 2.9. Analysis of IL & demonstration-based RL on assembling tasks. We control the difficulty of tasks by changing clearance configurations. Top: Behavior Cloning. Bottom: DAPG+PPO. 1x=default clearance (3mm for PegInsertionSide and 0.5mm for PlugCharger); 10x = 10 times default clearance.

Observation Mode	PegInsertionSide(1x)	PegInsertionSide(10x)	PlugCharger(1x)	PlugCharger(10x)
Point Cloud	0.00 ± 0.00	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01
RGBD	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Observation Mode	PegInsertionSide(1x)	PegInsertionSide(10x)	PlugCharger(1x)	PlugCharger(10x)
Point Cloud	0.01 ± 0.02	0.74 ± 0.10	0.01 ± 0.01	0.02 ± 0.02
RGBD	0.01 ± 0.01	0.05 ± 0.03	0.01 ± 0.01	0.29 ± 0.07

2.11.8 Network Architectures and Hyperparameters for IL & RL

In this section, we present the specific network architectures and algorithm hyperparameters used for Section 2.5.2. Here we define "state vector" as the concatenation of proprioceptive robot state information and task-specific goal information (if given).

For IMPALA [27], the visual backbone is similar to ResNet10 [41], with hidden size in all layers equal to 64, normalizations removed, and the first convolution layer modified to have kernel size 4 and stride 4. Features from the last layer are projected to a 384-dimensional vector before being concatenated with the state vector. For PointNet [91], the hidden layer sizes are [64, 128, 512] before maxpooling over the number-of-points dimension. We do not use spatial normalization layers, and we add layer normalization to the output of each intermediate layer before maxpooling.

For both BC and DAPG+PPO, we use learning rate of 3e-4 for training and optimize with Adam [58]. For DAPG+PPO, the actor and critic networks share the same visual backbone. In addition, when training point cloud-based policies, we initialize the linear layer right before the policy and value head output to be zero. We find this helpful for stabilizing point cloud-based policy learning. Hyperparameters are shown in Table 2.10.

Table 2.10. Hyperparameters for DAPG+PPO.

Hyperparameters	Value
Discount (γ)	0.95
λ in GAE	0.95
PPO clip range	0.2
Max KL	0.1
Gradient norm clipping	0.5
Entropy	0.0
Number of samples per PPO step	20000
Number of samples per minibatch	300
Number of critic warmup epochs	4
Number of PPO update epochs	2
Critic loss coefficient	0.5
Demonstration loss coefficient	$0.1 \cdot 0.995^N$
Recompute advantage after each PPO update epoch	True
Reward normalization	True
Advantage normalization	True
Only use critic loss to update visual backbone	True
Reset environment upon success during policy rollout	False

2.12 Comparison with Other Benchmarks for Robotic Manipulation

Table 2.11 compares ManiSkill2 with other existing benchmarks for robotic manipulation. ManiSkill2 features large-scale demonstrations for every task, a great variety of objects, multi-controller support and conversion of demonstration action spaces, and a focus on fully physically implemented grasp. We invested significant efforts to select, fix, and re-model objects and integrate them to our task families, generate demonstrations with fully physical grasping, and perform large-scale visual manipulation benchmarking. Through these processes, we carefully verify all of our tasks. The low success rates on many tasks demonstrate that our benchmark poses interesting and challenging problems for the community.

We are actively working on current limitations of ManiSkill2: photorealism, domain randomization, and scene-level variation. First, the simulation backend (SAPIEN) of ManiSkill2 supports a ray-tracing renderer (Kuafu). However, photorealism is usally achieved at the cost

Table 2.11. Comparison with other existing benchmarks for robotic manipulation. The information of each benchmark is based on its major focus. For example, all the simulation backends of these benchmarks can support physically implemented grasp, but some of them focus on high-level actions and thus use abstract grasp for benchmarking algorithms. *Multi-controller support* measures whether a benchmark has implemented multiple controllers and provided interfaces to select one for each task.

Benchmark	**ManiSkill2**	BEHAVIOR-1K[3]	Habitat 2.0[4]	IsaacGym[5]	ManipulaThor[6]	MetaWorld	Robosuite	RLBench	TDW[7]
Grasp implementation	Physical	Abstract	Abstract	Physical	Abstract	Physical	Physical	Abstract	Abstract
#Demo trajectories	>**30k**	-	-	-	-	Procedural	~2000	Procedural	-
Multi-controller support	**Yes**	Unknown	**Yes**	No	No	No	**Yes**	**Yes**	No
Visual RL/IL baselines	**Full**	Limited	**Full**	No	**Full**	No	No	No	No
#Object models	>2144*[8]	3324	YCB	-	150	80	10	28	112
#Scenes	-	**306**	105	-	30	-	-	-	105
Ray-tracing support	Kuafu	Omniverse	-	-	-	-	NVISII	-	Unity
Domain randomization	Partial[9]	Unknown	No	**Yes**	No	No	**Yes**	**Yes**	No
Rigid-body simulation	SAPIEN	Omniverse	Bullet	PhysX 5	Unity	Mujoco	Mujoco	V-REP	Unity
Soft-body simulation	Warp-MPM[10]	Omniverse	-	-	-	-	-	-	-

of speed. For example, BEHAVIOR-1K [63] can only achieve 60 FPS with Nvidia Omniverse and require high-end GPUs. In this work, we focus on physically realistic short-horizon and low-level visuomotor manipulation. We have experimentally supported ray-tracing rendering for use cases like generating data offline for supervised learning or fine-tuning a policy learned on non-photorealistic data. Second, we have included domain randomization for physical parameters and visual appearance for some of tasks. We will provide tutorials for users to customize domain randomization according to their use cases. Third, we are introducing mobile manipulation tasks similar to those in ManipularTHOR ArmPointNav [24] and Habitat 2.0 [109] but with physically implemented base movement and grasp. Those tasks will involve scene-level variations and demand advanced navigation abilities.

2.13 Conclusion

To summarize, ManiSkill2 is a unified and accessible benchmark for generic and generalizable manipulation skills, providing 20 manipulation task families, over 2000 objects, and over 4M demonstration frames. It features highly efficient rigid-body simulation and rendering system for sample collection to train RL agents, real-time MPM-based soft-body environments,

and versatile multi-controller conversion support. We have demonstrated its applications in benchmarking sense-plan-act and imitation/reinforcement learning algorithms, and we show that learned policies on ManiSkill2 have the potential to transfer to the real world.

Acknowledgement

Chapter 2, in full, is a reprint of the material published in the 2023 International Conference on Learning Representations (ICLR): "ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills" (Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, Hao Su). The dissertation author was the primary investigator and author of this paper.

Chapter 3

Conditioning on Trajectory Sketches for Robotic Task Generalization

Generalization remains one of the most important desiderata for robust robot learning systems. While recently proposed approaches show promise in generalization to novel objects, semantic concepts, or visual distribution shifts, generalization to new tasks remains challenging. For example, a language-conditioned policy trained on pick-and-place tasks will not be able to generalize to a folding task, even if the arm trajectory of folding is similar to pick-and-place. Our key insight is that this kind of generalization becomes feasible if we represent the task through rough trajectory sketches. We propose a policy conditioning method using such rough trajectory sketches, which we call *RT-Trajectory*, that is practical, easy to specify, and allows the policy to effectively perform new tasks that would otherwise be challenging to perform. We find that trajectory sketches strike a balance between being detailed enough to express low-level motion-centric guidance while being coarse enough to allow the learned policy to interpret the trajectory sketch in the context of situational visual observations. In addition, we show how trajectory sketches can provide a useful interface to communicate with robotic policies – they can be specified through simple human inputs like drawings or videos, or through automated methods such as modern image-generating or waypoint-generating methods. We evaluate *RT-Trajectory* at scale on a variety of real-world robotic tasks, and find that *RT-Trajectory* is able to perform a wider range of tasks compared to language-conditioned and goal-conditioned policies, when

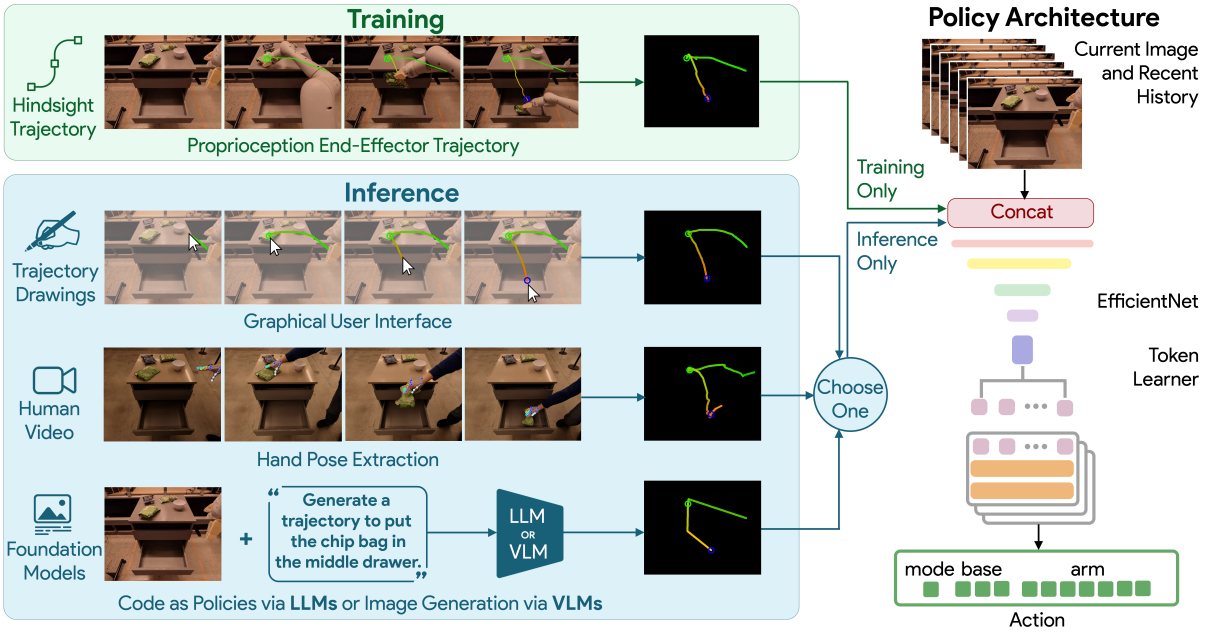


Figure 3.1. We propose *RT-Trajectory*, a framework for utilizing coarse trajectory sketches for policy conditioning. We train on hindsight trajectory sketches (top left) and evaluate on inference trajectories (bottom left) produced via *Trajectory Drawings*, *Human Videos*, or *Foundation Models*. These trajectory sketches are used as task specification for an RT-1 [12] policy backbone (right). The trajectories visually describe the end-effector motions (curves) and gripper interactions (circles).

provided the same training data. Evaluation videos can be found at https://rt-trajectory.github.io/.

3.1 Introduction

The pursuit of generalist robot policies has been a perennial challenge in robotics. The goal is to devise policies that not only perform well on known tasks but can also generalize to novel objects, scenes, and motions that are not represented in the training dataset. The generalization aspects of the policies are particularly important because of how impractical and prohibitive it is to compile a robotic dataset covering every conceivable object, scene, and motion. In this work we focus on the aspects of policy learning that, as we later show in the experiments, can have a large impact of their generalization capabilities: task specification and policy conditioning.

Traditional approaches to task specification include one-hot task conditioning [57], which

has limited generalization abilities since one-hot vector does not capture the similarities between different tasks. Recently, language conditioning significantly improves generalization to new language commands [12], but it suffers from the lack of specificity, which makes it difficult to generalize to a new motion that can be hard to describe. Goal image or video conditioning [69, 15], two other alternatives, offer the promise of more robust generalization and can capture nuances hard to express verbally but easy to show visually. However, it has been shown to be hard to learn from [50] and requires more effort to provide at test time, making it less practical. Most importantly, policy conditioning not only impacts the practicality of task specification, but can have a large impact on generalization at inference time. If the representation of the task is similar to the one of the training tasks, the underlying model is more likely able to interpolate between these data points. This is often reflected with the type of generalization exhibited in different conditioning mechanisms – for example, if the policy is conditioned on natural language commands, it is likely to generalize to a new phrasing of the text command, whereas that same policy when trained on pick-and-place tasks will struggle with generalizing to a folding task, even if the arm trajectory of folding is similar to pick-and-place, because in language space, this new task is outside of the previously seen data. This begs a question: can we design a better conditioning modality that is expressive, practical and, at the same time, leads to better generalization to new tasks?

To this end, we propose to use a *coarse* trajectory as a middle-ground solution between expressiveness and ease of use. Specifically, we introduce the use of a 2D trajectory projected into the camera's field of view, assuming a calibrated camera setup. This approach offers several advantages. For example, given a dataset of demonstrations, we can automatically extract hindsight 2D trajectory labels without the need for manual annotation. In addition, trajectory labels allow us to explicitly reflect similarities between different motions of the robot, which, as we show in the experiments, leads to better utilization of the training dataset resulting in a wider range of tasks compared to language- and goal-conditioned alternatives. Furthermore, humans or modern image-editing models can sketch these trajectories directly onto an image, making it a

simple yet expressive policy interface.

The main contribution of this paper is a novel policy conditioning framework *RT-Trajectory* that fosters task generalization. This approach employs 2D trajectories as a human-interpretable yet richly expressive conditioning signal for robot policies. Our experimental setup involves a variety of object manipulation tasks with both known and novel objects. Our experiments show that *RT-Trajectory* outperforms existing policy conditioning techniques, particularly in terms of generalization to novel motions, an open challenge in robotics.

3.2 Related Work

In this section, we discuss prior works studying generalization in robot learning as well as works proposing specific policy conditioning representations.

Generalization in Robot Learning

Recent works have studied how learning-based robot policies may generalize robustly to novel situations beyond the exact data seen during training. Empirical studies have analyzed generalization challenges in robotic imitation learning, focusing on 2D control [113], demonstration quality [76], visual distribution shifts [130], and action consistency [6]. In addition, prior works have proposed evaluation protocols explicitly testing policy generalization; these include generalizing to novel semantic attributes [102], holdout language templates [50], unseen object categories [88, 73, 103, 106], new backgrounds and distractors [17, 134], combinations of distribution shifts [12, 53], open-set language instructions [129, 47], and web-scale semantic concepts [11]. While these prior works largely address semantic and visual generalization, we additionally study task generalization which include situations which require combining seen states and actions in new ways, or generalizing to wholly unseen states or motions altogether.

Policy Conditioning Representations

We examine a few approaches for policy conditioning. Broadly, there are 2 axes to consider: (1) over-specification and under-specification of goals, and (2) conditioning on all

states in a trajectory versus only the end state. The most prolific recent body of work focuses on language-conditioned policies [50, 12, 11, 83, 1, 42, 70], which utilize templated or freeform language as task specification. Language-conditioned policies can be thought of as *under-specified on the end state* (e.g. there are many possible end-states for a policy that completes `pick can`). There are many image-conditioned policy representations with the most popular technique being goal-image conditioning: where a final goal image defines the desired task's end-state [9, 69]. Goal image conditioned policies can be thought of as *over-specified on the end state* (i.e. "what to do") because they define an entire configuration, some of which might not be relevant. For example, the background pixels of the goal image might not be pertinent to the task, and instead contain superfluous information. There are some examples of intermediate levels of specification that propose 2D and 3D object-centric representations [106, 102, 47], using a multimodal embedding that represents the task as a joint space of task-conditioned text and goal-conditioned image [129, 53, 102], and describing the policy as code [64] which constrains how to execute every state. An even more detailed type of state-specification would be conditioning on an entire RGB video which is equivalent to *over-specification over the entire trajectory of states* (i.e. "how to do it") [15]. However, encoding long videos in-context is challenging to scale, and learning from high-dimensional videos is a challenging learning problem [50]. In contrast, our approach uses a lightweight coarse level of state-specification, which aims to strike a balance between sufficient state-specification capacity to capture salient state properties while still being tractable to learn from. We specifically compare against language-conditioning and goal-image conditioning baselines, and show the benefits of using a mid-level conditioning representation such as coarse trajectory sketches.

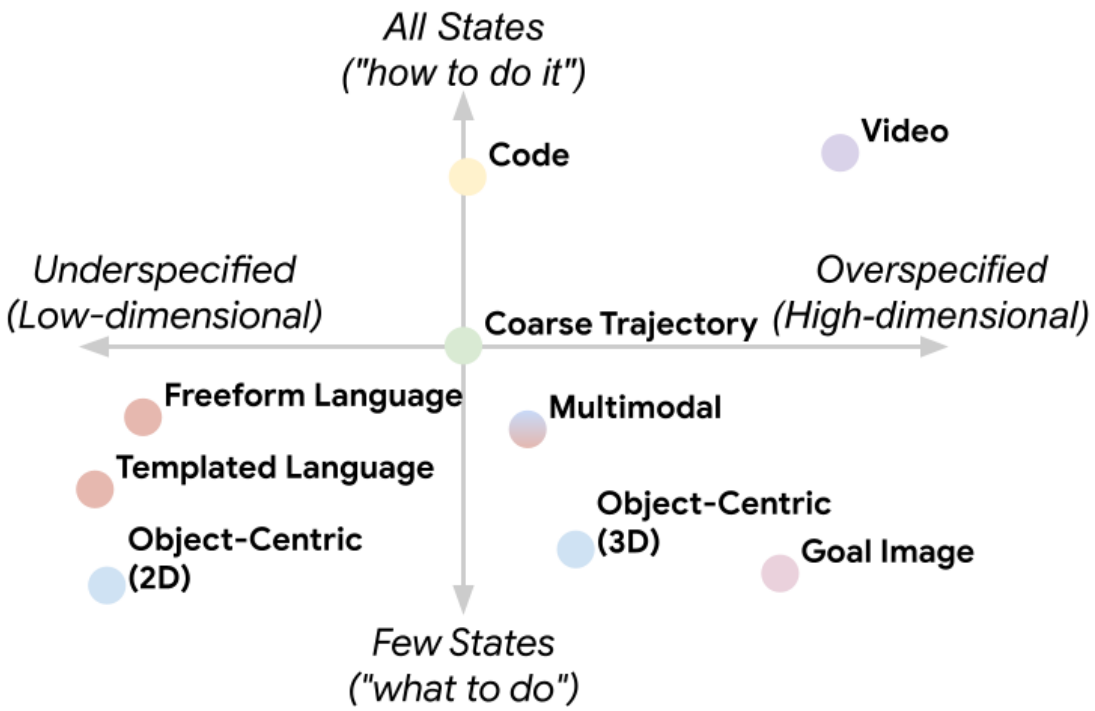


Figure 3.2. The choice of robot policy representation balances specification detail and focusing policies on "what to do" compared with "how to do it".

3.3 Method

3.3.1 Overview

Our goal is to learn a robotic control policy that is able to utilize a 2D coarse trajectory sketch image as its conditioning. A system diagram for our proposed approach can be seen in Fig 3.1. During policy training, we first perform hindsight trajectory labeling to obtain trajectory conditioning labels from the demonstration dataset (Section 3.3.2). This enables us to re-use existing demonstration dataset and ensures the scalability of our proposed approach to new datasets. We then train a transformer-based control policy that is conditioned on the 2D trajectory sketches using imitation learning (Section 3.3.3). During inference time, the user or a high-level planner is presented an initial image observation from the robot camera, and creates a rough 2D trajectory sketch that specifies the desired motion (Fig. 3.1 bottom left), which is then fed into the trained control policy to perform the designated manipulation task.

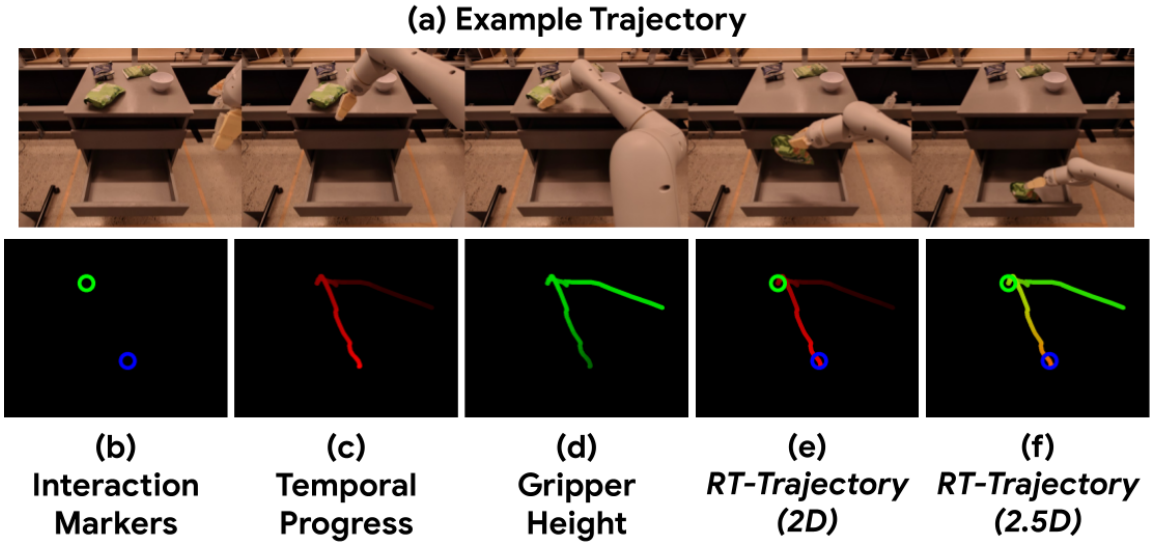


Figure 3.3. Visualization of the two hindsight trajectory sketch representations we study. Given (a) an example robot trajectory, we extract (b) gripper interaction markers, (c) temporal progress along the 2D end-effector waypoints, and (d) end-effector height. Combining (b) and (c) results in (e) ***RT-Trajectory (2D)***, while combining (b), (c), and (d) results in (f) ***RT-Trajectory (2.5D)***.

3.3.2 Hindsight Trajectory Labels

In this section, we describe how we acquire training trajectory conditioning labels from the demonstration dataset. We introduce three basic elements for constructing the trajectory representation format: 2D Trajectories, Color Grading, and Interaction Markers.

2D Trajectory

For each episode in the demonstration dataset, we extract a 2D trajectory of robot end-effector center points. Concretely, given the proprioceptive information recorded in the episode, we obtain the 3D position of the robot end-effector center defined in the robot base frame at each time step, and project it to the camera space given the known camera extrinsic and intrinsic parameters. We assume that the robot base and camera do not move within the episode, which is common for stationary manipulation. Given a 2D trajectory (a sequence of pixel positions), we draw a curve on a blank image, by connecting 2D robot end-effector center points at adjacent time steps through straight lines.

Color Grading

To express relative temporal motion, which encodes such as velocity and direction, we also explore using the red channel of the trajectory image to specify the normalized time step $\frac{t+1}{T}$, where t is the current time step and T is the total episode length. Additionally, we propose incorporating height information into the trajectory representation by utilizing the green channel of the trajectory image to encode normalized height relative to the robot base $\frac{h_{t+1}-h_{min}}{h_{max}-h_{min}}$.

Interaction Markers

For robot manipulation tasks, time steps when the end-effector interacts with the environment are particularly important. Thus, we explore visual markers that explicitly highlight the time steps when the gripper begins to grasp and release objects. Concretely, we first compute whether the gripper has contact with objects by checking the difference $\delta_t = \hat{p}_t - p_t$ between the sensed (p_t) and target ($\hat{p}_t$) gripper joint positions. If the difference $\delta_t > 0$ and $\hat{p}_t > \epsilon$, where ϵ is a threshold of closing action (p_t increases as the gripper closes), it indicates that the gripper is closing and grasping certain object. If the status change, e.g., $\delta_t < 0 \vee \hat{p}_t \leq \epsilon$ but $\delta_{t+1} > 0 \wedge \hat{p}_{t+1} > \epsilon$, we consider the time step t as a key step for the closing action. Similarly, we can find the key time steps for the opening action. We draw green (or blue) circles at the 2D robot end-effector center points of all key time steps for closing (or opening) the gripper.

Trajectory Representations

In this work, we propose two forms of trajectory representation from different combinations of the basic elements. In the first one, ***RT-Trajectory (2D)***, we construct an RGB image containing the 2D Trajectory with temporal information and Interaction Markers to indicate particular robot interactions (Fig. 3.3 (e)). In the second representation, we introduce a more detailed trajectory representation ***RT-Trajectory (2.5D)***, which includes the height information in the 2D trajectory (Fig. 3.3 (f)).

3.3.3 Policy Training

We leverage Imitation Learning due to its strong success in multitask robotic imitation learning settings [50, 9]. More specifically, we assume access to a collection of successful robot demonstration episodes. Each episode τ contains a sequence of pairs of observations o_t and actions a_t: $\tau = \{(o_t, a_t)\}$. The observations include RGB images obtained from the head camera x_t and hindsight trajectory sketch c_{traj}. We then learn a policy π represented by a Transformer [115] using Behavior Cloning [89] following the RT-1 framework [12], by minimizing the log-likelihood of predicted actions a_t given the input image and trajectory sketch. To support trajectory conditioning, we modify the RT-1 architecture as follows. The trajectory sketch is concatenated with each RGB image along the feature dimension in the input sequence (a history of 6 images), which is processed by the image tokenizer (an ImageNet pretrained EfficientNet-B3). For the additional input channels to the image tokenizer, we initialize the new weights in the first convolution layer with all zeros. Since the language instruction is not used, we remove the FiLM layers used in the original RT-1.

3.3.4 Trajectory Conditioning during Inference

During inference, a trajectory sketch is required to condition *RT-Trajectory*. We study 4 different methods to generate trajectory sketches: *human drawings*, *human videos*, *prompting LLMs with Code as Policies*, and *image generation models*.

Human-drawn Sketches

Human-drawn sketches are an intuitive and practical way for generating trajectory sketches. To scalably produce these sketches, we design a simple graphical user interface (GUI) for users to draw trajectory sketches given the robot's initial camera image, as shown in Sec. 3.5.1.

Human Demonstration Videos with Hand-object Interaction

First-person human demonstration videos are an alternative input. We estimate the trajectory of human hand poses from the video, and convert it to a trajectory of robot end-effector

poses, which can later be used to generate a trajectory sketch.

Prompting LLMs with Code as Policies

Large Language Models have demonstrated the ability to write code to perform robotics tasks [64]. We follow a similar recipe as described in [35] to build a prompt which contains text descriptions about the objects in the scene detected by a VLM, the robot constraints, the gripper orientations and coordinate systems, as well as the task instruction. By using this prompt, the LLM writes code to generate a series of 3D poses - originally intended to be executed with a motion planner, which we can then re-purpose to draw the trajectory sketch on the initial image to condition *RT-Trajectory*.

Image Generation Models

Since our trajectory conditioning is represented as an image, we can use text-guided image generation models to generate a trajectory sketch provided the initial image and language instruction which describes the task. In our work, we use a PaLM-E style [23] model that generates vector-quantized tokens derived from ViT-VQGAN [132] that represent the trajectory image. Once detokenized, the resulting image can be used to condition *RT-Trajectory*.

3.4 Experiments

Our real robot experiments aim to study the following questions:

1. Can *RT-Trajectory* generalize to tasks beyond those contained in the training dataset?
2. Can *RT-Trajectory* trained on hindsight trajectory sketches generalize to diverse human-specified or automated trajectory generation methods at test time?
3. What emergent capabilities are enabled by *RT-Trajectory*?
4. Can we quantitatively measure how dissimilar evaluation trajectory motions are from training dataset motions?

3.4.1 Experimental Setup

We use a mobile manipulator robot from Everyday Robots in our experiments, which has a 7 degree-of-freedom arm, a two-fingered gripper, and a mobile base.

Seen Skills

We use the RT-1 [12] demonstration dataset for training. The language instructions consist of 8 different manipulation skills (e.g., `Move Near`) operating on a set of 17 household kitchen items; in total, the dataset consists of about 73K real robot demonstrations across 542 seen tasks, which were collected by manual teleoperation. A more detailed overview is shown in Table 3.1.

Table 3.1. The list of seen training tasks with their descriptions and example language instructions. Language instructions are only used for language-conditioned baselines. "Count" refers to the number of distinct tasks per skill (e.g., `Pick coke can` and `Pick apple` are two different tasks).

Skill	Count	Description	Example Instruction
Pick `Object`	17	Lift the object off the surface	pick coke can
Move `Object` Near `Object`	337	Move the first object near the second	move pepsi can near rxbar blueberry
Place `Object` Upright	8	Place an elongated object upright	place water bottle upright
Knock `Object` Over	8	Knock an elongated object over	knock redbull can over
Open `Drawer`	3	Open any of the cabinet drawers	open the top drawer
Close `Drawer`	3	Close any of the cabinet drawers	close the middle drawer
Place `Object` into `Receptacle`	84	Place an object into a receptacle	place brown chip bag into white bowl
Pick `Object` from `Receptacle` and Place on the Counter	82	Pick an object up from a location and then place it on the counter	pick green jalapeno chip bag from paper bowl and place on counter
Total	542		

Unseen Skills

We propose 7 new evaluation skills which include unseen objects and manipulation workspaces, as shown in Table 3.2 and Fig. 3.4. Both `Upright and Move` and `Move within Drawer` examine whether the policy can combine different seen skills to form a new one. For example, `Move within Drawer` studies whether the policy is able to move objects within the drawer while the seen skill `Move Near` only covers those motions at height of the tabletop. `Restock Drawer` requires the robot to place snacks into the drawer at an empty slot. It studies whether the policy is able to place objects at target positions precisely. `Place Fruit` inspects whether the policy can

place objects into unseen containers. `Pick from Chair` investigates whether the policy can pick objects at an unseen height in an unseen manipulation workspace. `Fold Towel` and `Swivel Chair` showcase the capability to manipulate a deformable object and interact with an underactuated system.

Table 3.2. The list of unseen evaluation tasks with their descriptions and example language instructions. Language instructions are only used for language-conditioned baselines. "Count" refers to the number of scenes collected for evaluation.

Skill	Count	Description	Example instruction
Place Fruit	12	Place fruit into the container	place orange into basket
Upright and Move	6	Place an object upright *and* move it near another	place green can upright near pepsi can
Move within Drawer	6	Move one object near another *within* the drawer	move coke can near 7up can at top drawer
Restock Drawer	12	Place objects into the desired position in the drawer	place coke can into the top right of top drawer
Pick from Chair	8	Pick an object placed on the chair	pick apple from chair
Fold Towel	4	Fold the towel by moving one corner to another	fold towel from bottom right
Swivel Chair	10	Swivel the office chair	push the chair

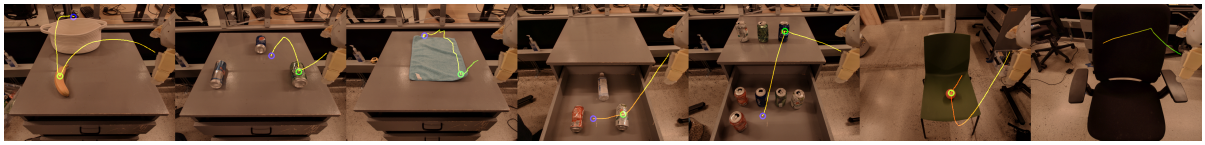


Figure 3.4. Visualization of trajectory sketches overlaid on the initial image for 7 unseen skills. From left to right: `Place Fruit`, `Upright and Move`, `Fold Towel`, `Move within Drawer`, `Restock Drawer`, `Pick from Chair`, `Swivel Chair`. See the rollouts in Fig. 3.18.

Evaluation Protocol

Different trajectory sketches will prompt *RT-Trajectory* to behave differently. To make the quantitative comparison between different methods as fair as possible, we propose the following evaluation protocol. For each skill to evaluate, we collect a set of *scenes*. Each scene defines the initial state of the task, described by an RGB image taken by the robot head camera. During evaluation, we first align relevant objects to their original arrangements in the *scene*, and then run the policy. For conditioning *RT-Trajectory*, we use human drawn sketches for unseen tasks in Sec. 3.4.2. In Sec. 3.4.3, we evaluate other trajectory sketch generation methods described in Sec. 3.3.4.

3.4.2 Unseen Task Generalization

In this section, we compare *RT-Trajectory* with other learning-based baselines on generalization to the unseen task scenarios introduced in Sec 3.4.1.

- RT-1 [12]: language-conditioned policy trained on the same training data;
- RT-2 [11]: language-conditioned policy trained on a mixture of our training data and internet-scale VQA data;
- RT-1-Goal: goal-conditioned policy trained on the same training data.

For *RT-Trajectory*, we manually generate trajectory sketches via the GUI (see Sec. 3.5.1). Details about trajectory generation are described in Sec. 3.5.2. For *RT-1-Goal*, implementation details and goal conditioning generation are presented in Sec. 3.5.4. The results are shown in Fig. 3.5 and Table 3.3. The overall success rates of our methods, *RT-Trajectory (2D)* and *RT-Trajectory (2.5D)*, are 50% and 67% respectively, which outperform our baselines by a large margin: *RT-1* (16.7%), *RT-2* (11.1%), *RT-1-Goal* (26%). Language-conditioned policies struggle to generalize to the new tasks with semantically unseen language instructions, even if motions to achieve these tasks were seen during training (see Sec. 3.4.5). *RT-1-Goal* shows better generalization than its language-conditioned counterparts. However, goal conditioning is much harder to acquire than trajectory sketches during inference in new scenes and is sensitive to task-irrelevant factors (e.g., backgrounds). *RT-Trajectory (2.5D)* outperforms *RT-Trajectory (2D)* on the tasks where height information helps reduce ambiguity. For example, with 2D trajectories only, it is difficult for *RT-Trajectory (2D)* to infer correct picking height, which is critical for `Pick from Chair`.

3.4.3 Diverse Trajectory Generation Methods

In this section, we aim to study whether *RT-Trajectory* is able to generalize to trajectories from more automated and general processes at inference time. Specifically, we evaluate

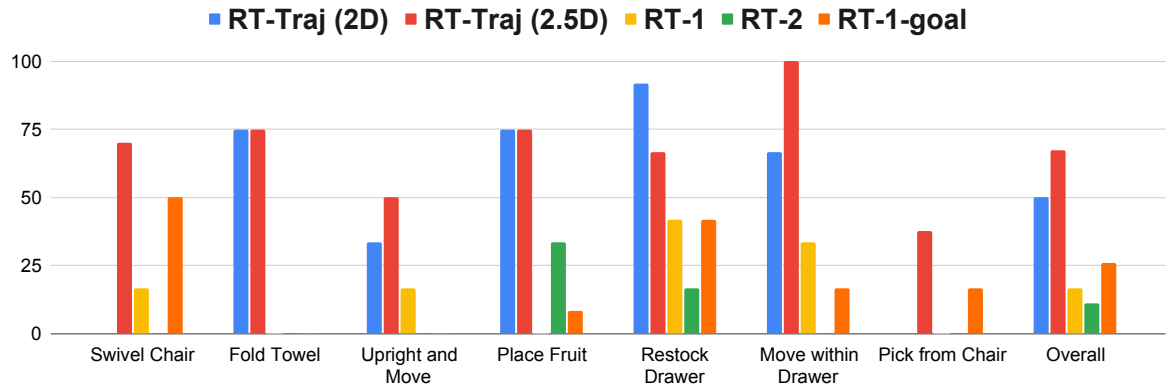


Figure 3.5. Success rates for unseen tasks when conditioning with human drawn sketches. Scenarios contain a variety of difficult settings which require combining seen motions in novel ways or generalizing to new motions. Each policy is evaluated for a total of 64 trials across 7 different scenarios.

Table 3.3. Success rates for unseen tasks when conditioning with human drawn sketches.

Task	RT-Traj (2D)	RT-Traj (2.5D)	RT-1	RT-2	RT-1-goal
`Place Fruit`	75%	75%	0%	33%	8%
`Upright and Move`	33%	50%	17%	0%	0%
`Move within Drawer`	67%	100%	33%	0%	17%
`Restock Drawer`	92%	67%	42%	17%	42%
`Pick from Chair`	0%	38%	0%	0%	17%
`Fold Towel`	75%	75%	0%	0%	0%
`Swivel Chair`	0%	70%	17%	0%	50%
Overall	50%	67%	17%	11%	26%

quantitatively how *RT-Trajectory* performs when conditioned on coarse trajectory sketches generated by *human video demonstrations*, LLMs via *Prompting with Code as Policies*, and show qualitative results for *image generating VLMs*. Additionally, we compare *RT-Trajectory* against a non-learning baseline (*IK Planner*) to follow the generated trajectories: an inverse-kinematic (IK) solver is applied to convert the end-effector poses to joint positions, which are then executed by the robot.

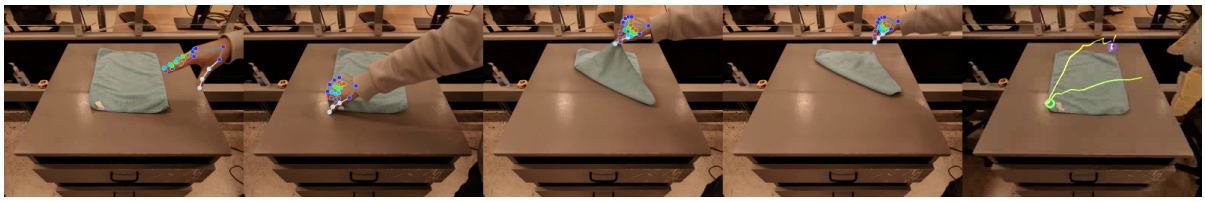


Figure 3.6. Trajectory from human demonstration video to fold a towel. From left to right, the first 4 images show the human demonstration, and the last image shows the derived trajectory sketch.

Table 3.4. Success rate of different trajectory generation approaches across tasks.

(a) Trajectory from human video demonstrations.

Method	Pick	Fold Towel
IK Planner	42%	25%
Ours (2D)	94%	75%
Ours (2.5D)	100%	75%

(b) Trajectory from LLM prompting.

Method	Pick	Open Drawer
IK Planner	83%	71%
Ours (2D)	89%	60%
Ours (2.5D)	89%	60%

Human Demonstration Videos

We collect 18 and 4 first-person human demonstration videos with hand-object interaction for `Pick` (seen training skill) and `Fold Towel`. An example is shown in Fig. 3.6. Details about video collection and how trajectory sketches are derived from videos are described in Sec. 3.5.3. The resulting trajectory sketches are more squiggly than the ones for training. Results are shown in Table 3.4a.

Prompting with Code as Policies

We prompt an LLM [85] to write code to generate trajectories given the task instructions and object labels for two seen skills, `Pick` and `Open Drawer`. After executing the code written by the LLM, we get a sequence of target robot waypoints which can then be processed into a trajectory sketch. In contrast with human-specified trajectories, LLM-generated trajectories are designed to be executed by an IK planner and are therefore precise and linear as seen in Fig. 3.9. While they are also different from the hindsight trajectories in the training data, *RT-Trajectory* is able to execute them correctly and outperform the IK planner in diverse pick tasks due to its ability to adapt motion to the scene nuances like object orientation. Results are shown in Table

3.4b.

Image Generation Models

We condition the VLM with a language instruction and an initial image to output trajectory tokens which are de-tokenized into 2D pixel coordinates for drawing the trajectory. Qualitative examples are shown in Fig 3.7. Although we see that generated trajectory sketches are noisy and quite different from the training hindsight trajectory sketches, we find promising signs that *RT-Trajectory* still performs reasonably. As image-generating VLMs rapidly improve, we expect that their trajectory sketch generating capabilities will improve naturally in the future and be usable by *RT-Trajectory*.

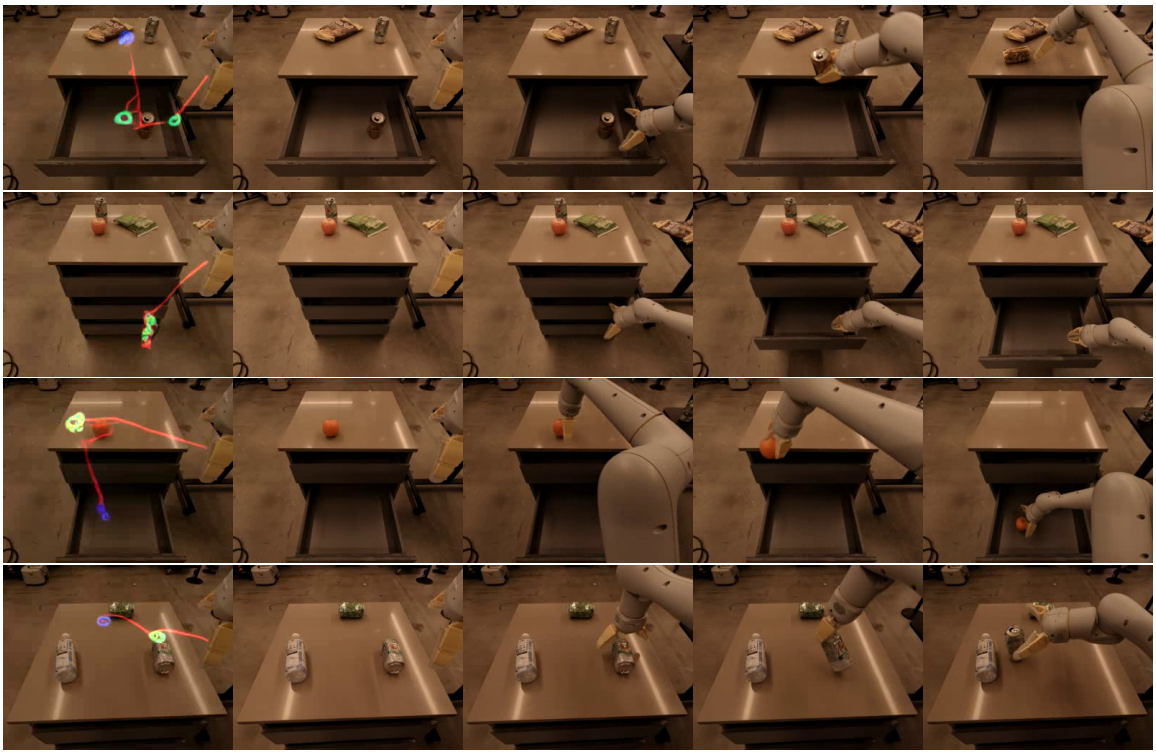


Figure 3.7. Example trajectories from image generation models. Each row shows the trajectory sketch overlaid on the first frame and the rollout. The language instructions are: `pick orange can from top drawer and place on counter`, `open middle drawer`, `place orange into middle drawer`, `move 7up can near blue plastic bottle`.

3.4.4 Emergent Capabilities and Behaviors

Prompt Engineering for Robot Policies

Similar to how LLMs respond differently in response to language prompt engineering, *RT-Trajectory* enables *visual* prompt engineering, where a trajectory-conditioned policy may exhibit better performance when the initial scene is fixed but the coarse trajectory prompts are improved. We find that changing trajectory sketches induces *RT-Trajectory* to change behavior modes in a reproducible manner, which suggests an intriguing opportunity: if a trajectory-conditioned robot policy fails in some scenario, a practitioner may just need to "query the robot" with a different trajectory prompt, as opposed to re-training the policy or collecting more data. Qualitatively, this is quite different from standard development practices with language-conditioned robot policies, and may be viewed as an early exploration into zero-shot instruction tuning for robotic manipulation, similar to capabilities seen in language modeling [13].

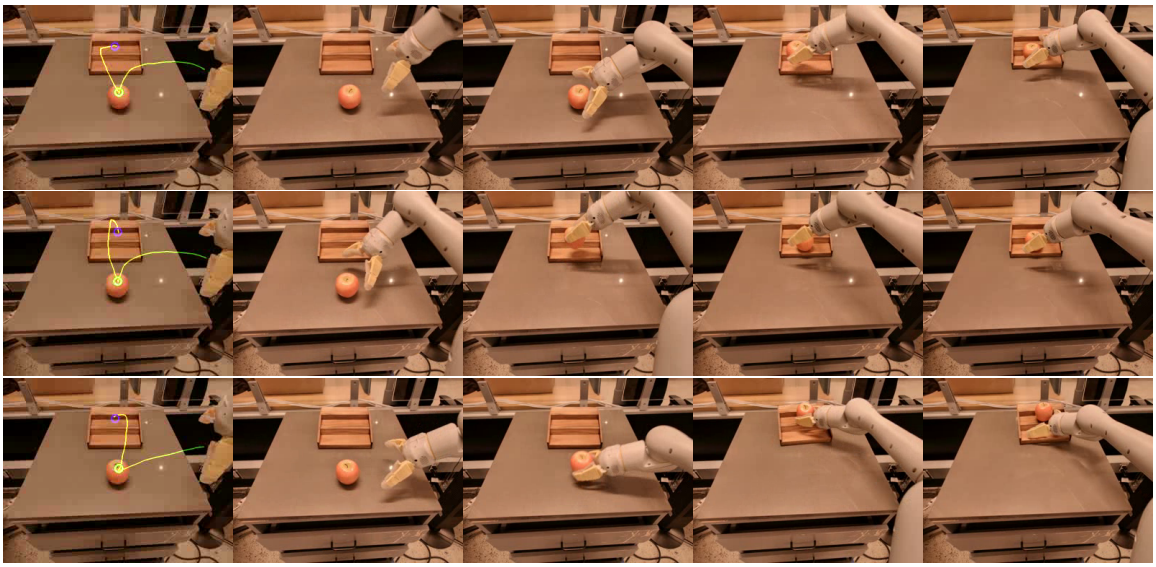
Fig. 3.8 illustrates two examples of prompt engineering. For instance, if the user wants to prompt *RT-Trajectory* to place an object at a high position, it is better to draw a trajectory that first reaches a higher peak, and then move downward to the target.

Retry behavior

Compared to non-learning methods, *RT-Trajectory* is able to recover from execution failures. Fig. 3.9 illustrates the retry behavior emerged when *RT-Trajectory* is opening the drawer given the trajectory sketch generated by prompting LLMs with Code as Policies (CaP) mentioned in Sec. 3.3.4. After a failure attempt to open the drawer by its handle, the robot retried to grasp the edge of the drawer, and managed to pull the drawer.

Height-aware Disambiguation for *RT-Trajectory (2.5D)*

2D trajectories (without depth information) are visually ambiguous for distinguishing whether the robot should move its arm to a deeper or higher. We find that height-aware color grading for *RT-Trajectory (2.5D)* can effectively help reduce such ambiguity, as shown in Fig. 3.10.



(a) The objective is to place the apple onto the *middle* stage.

(b) The objective is to place the apple onto the *top* stage.

Figure 3.8. Case studies in prompt engineering. Each row shows the trajectory sketch overlaid on the first frame and the corresponding rollout. As seen in the first two rows, suboptimal trajectory prompts result in failures. However, by keeping the initial scene conditions identical but simply improving the trajectory prompt, the policy is able to succeed.

Generalizing to Realistic Settings

Prior works studying robotic generalization often evaluate only a few distribution shifts at once, since generalizing to simultaneous physical and visual variations is challenging; however, these types of simultaneous distribution shifts are widely prevalent in real world settings. As a

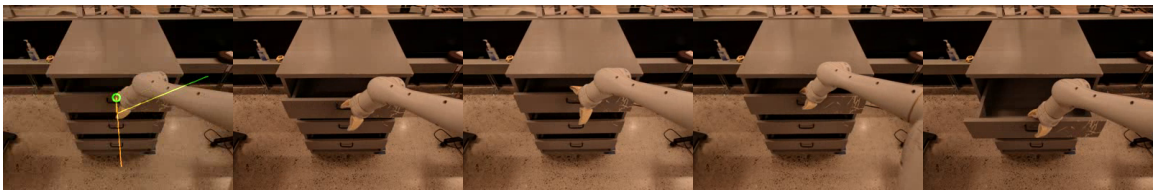


Figure 3.9. Example of retry behavior. The first image is the trajectory sketch generated from the CaP overlaid on the initial observation. The remaining images show the rollout. The robot first attempts to open the drawer by grasping its handle, but fails (2nd image). Then, it retries to open the drawer by grasping the edge instead.

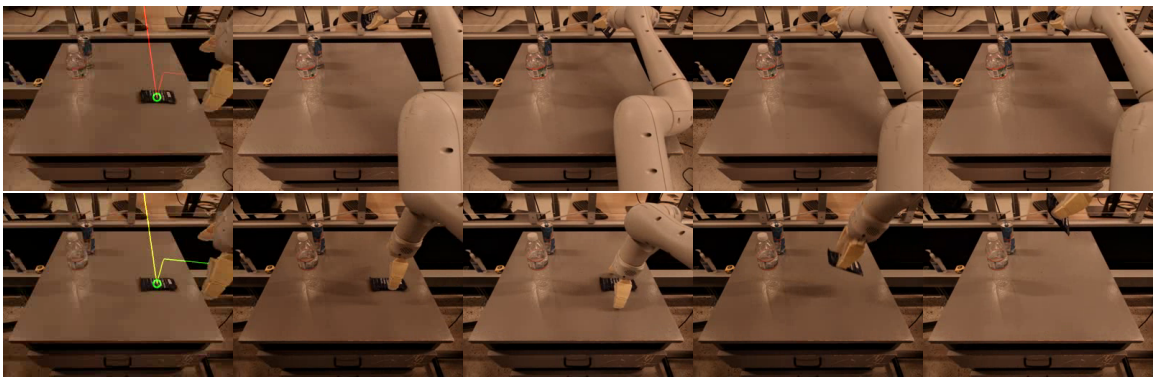


Figure 3.10. Comparison between *RT-Trajectory (2D)* and *RT-Trajectory (2.5D)*. Given the same 2D trajectory generated by the CaP, *RT-Trajectory (2.5D)* lifts the object while *RT-Trajectory (2D)* moves the object to a deeper position due to the ambiguity of a 2D trajectory.

qualitative case study, we evaluate *RT-Trajectory* in 2 new buildings in 4 realistic novel rooms which contain entirely new backgrounds, lighting conditions, objects, layouts, and furniture geometries. With little to moderate trajectory prompt engineering, we find that *RT-Trajectory* is able to successfully perform a variety of tasks requiring novel motion generalization and robustness to out-of-distribution visual distribution shifts. These tasks are visualized in Fig. 3.11 and rollouts are shown fully in Fig. 3.19.

3.4.5 Measuring Motion Generalization

We wish to explicitly measure motion similarity in order to better understand how *RT-Trajectory* is able to generalize to unseen scenarios and how well it can tackle the challenges of novel motion generalization. Towards this, we intend to compare evaluation trajectories to the most similar trajectories seen during training. To measure the distance between two end-effector

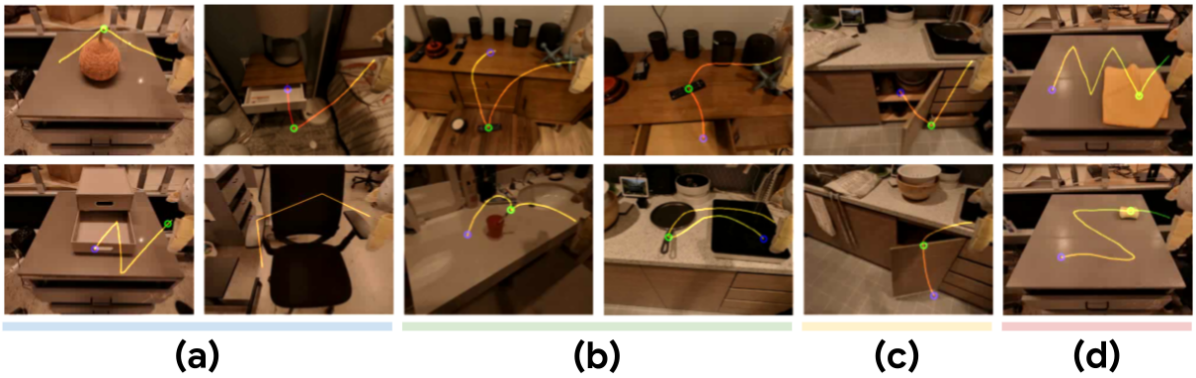


Figure 3.11. Example *RT-Trajectory* evaluations in realistic scenarios involving (a) novel articulated objects requiring new motions, (b) manipulation on new surfaces in new buildings in new heights, (c) interacting with a pivot-hinge cabinet despite training only on sliding-hinge drawers, and (d) circuitous tabletop patterns extending beyond direct paths in the training dataset. Full rollouts are shown in Fig. 3.19 and the supplemental video at https://rt-trajectory.github.io/.

motion trajectories, we employ the Fréchet distance [29, 25], a measure that quantifies the similarity between two curves by finding the minimum "leash length" required for two agents traversing each curve simultaneously while maintaining their respective temporal order. As a well-adopted similarity measure in computer vision and vehicle tracking [8], Fréchet distance may be a reasonable choice for comparing 3D robot end-effector waypoint trajectories since it is order-preserving and parameterization independent [43].

Specifically, consider two trajectories τ and τ' where each trajectory contains n waypoints $\tau = \{\rho_0, \rho_1, ..., \rho_m\}$ and $\tau' = \{\rho'_0, \rho'_1, ..., \rho'_n\}$, and $d(\rho_i, \rho'_i)$ is a distance measure like Euclidean distance. Then, using the notation $\tau[1{:}]$ to denote removing the first element and returning the rest of the sequence τ, the Fréchet distance F_D is recursively defined as:

$$F_D(\tau, \tau') = \max\left(d(\rho_0, \rho'_0), \min\left\{F_D(\tau[1{:}], \tau'[1{:}]), F_D(\tau, \tau'[1{:}]), F_D(\tau[1{:}], \tau')\right\}\right)$$

In this work, each waypoint is the sensed end-effector center position and the distance measure is Euclidean distance. Note that the orientation and interaction (closing/opening action) are not taken into consideration.

By computing the distance between a query trajectory and all trajectories in our training

dataset, we can retrieve the most similar trajectories our policy has been trained on. We perform this lookup for trajectories from the rollouts for the unseen task evaluations in Sec. 3.3.4. Fig. 3.12 showcases the 10 most similar training trajectories for a selection of query trajectories.

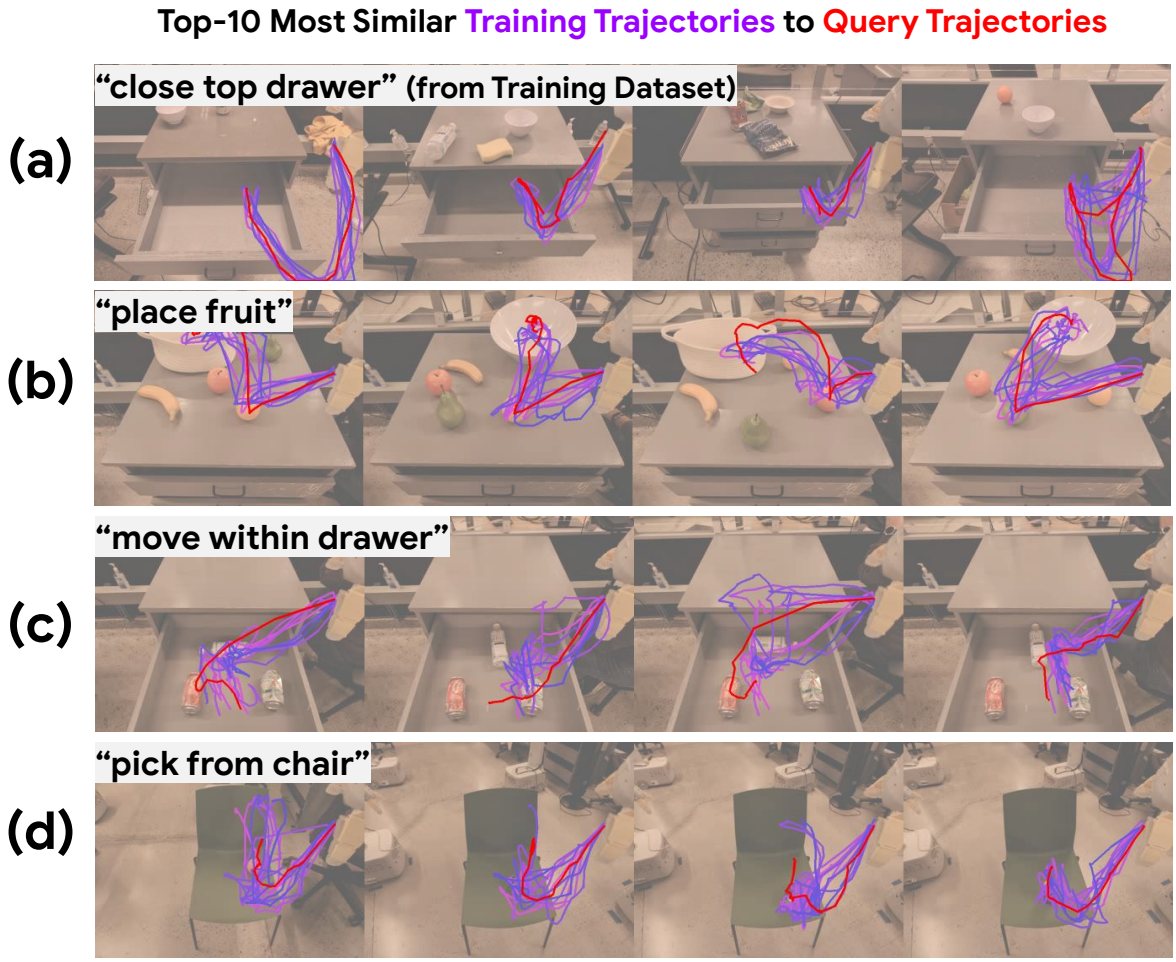


Figure 3.12. Each row contains 4 instances of an initial image of an evaluation rollout super-imposed with the executed evaluation trajectory (red) compared with the 10 most similar trajectories (purple) in the training dataset. Row (a) shows query trajectories of the in-distribution `close top drawer` skill seen in the training data. Rows (b,c,d) show query trajectories of unseen evaluation skills.

Fig. 3.13, 3.14, and 3.15 furthermore show statistics of the most similar training samples, such as the distribution of skill semantics. We find that the trajectories for unseen tasks show varying levels of similarity to training trajectories. For example, the motion for `place a fruit into a tall bowl` may be surprisingly similar to the motion for particular seen instances of the

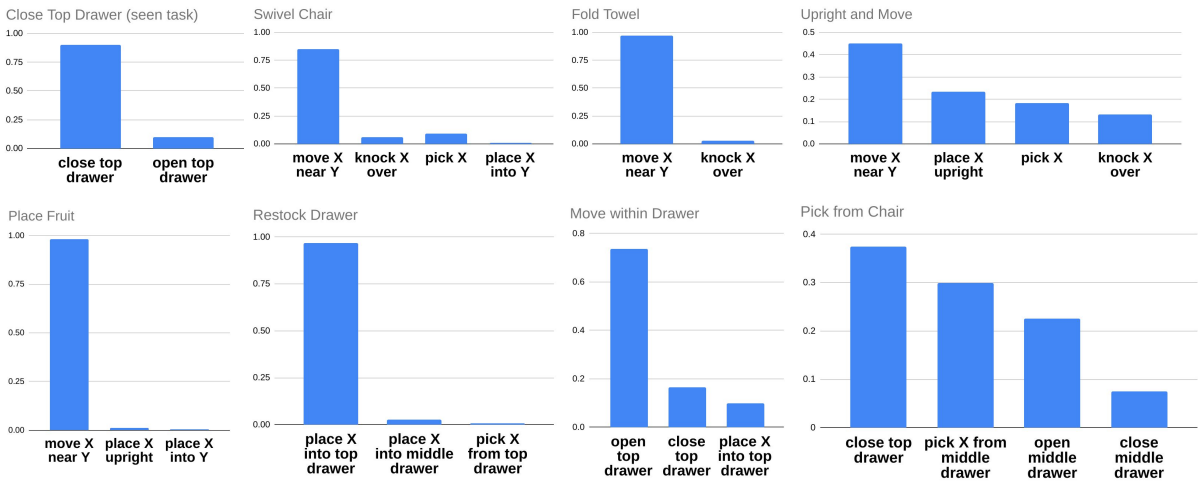


Figure 3.13. Semantic relevance measures how the semantic skills of rollout trajectories compare to the semantic skills of the most similar training trajectories, as measured by motion similarity. For the seen skill (`close top drawer`), the most similar training trajectories are largely of the same semantic skill. For the unseen skills, the most similar training trajectories are composed of semantically different tasks.

`move X near Y`. However, for many unseen skills, the most similar examples in the training data are still significantly more different than for examples within the training set. In addition, even for evaluation trajectories that seem close in shape to the most similar training trajectories, we find differences in precision-critical factors like the z-height of gripper interactions (picks that are just a few centimeter incorrect will not succeed) or semantic relevance (the most similar training trajectories describe different skills than the target trajectory). Thus, we expect that the proposed new skills for evaluation indeed require a mix of interpolating seen motions along with generalizing to novel motions altogether.

Figure 3.16 shows additional examples of evaluation trajectories and their most similar trajectories in the training dataset.

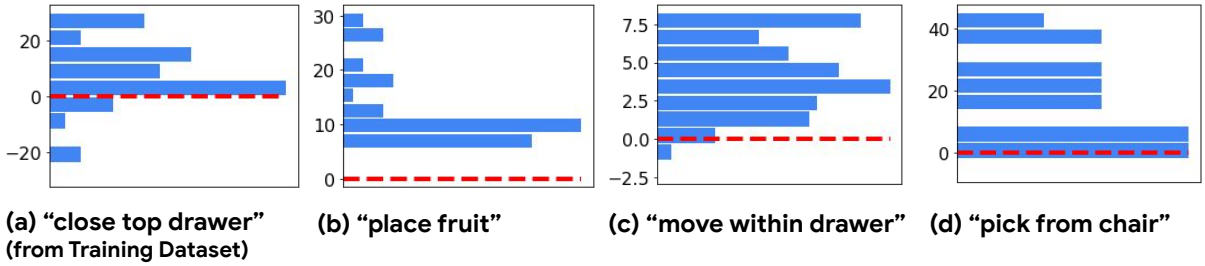


Figure 3.14. First-interaction height alignment compares the relative difference between the z-height of the first gripper interactions of query trajectories to the first gripper interactions of the most similar training trajectories, as measured by motion similarity. The red line represents the baseline relative difference of the query trajectory compared with itself, which would be a difference of 0.0. The unseen skills in general see large variance in the difference of first-interaction heights of the query trajectories compared to the most similar training trajectories.

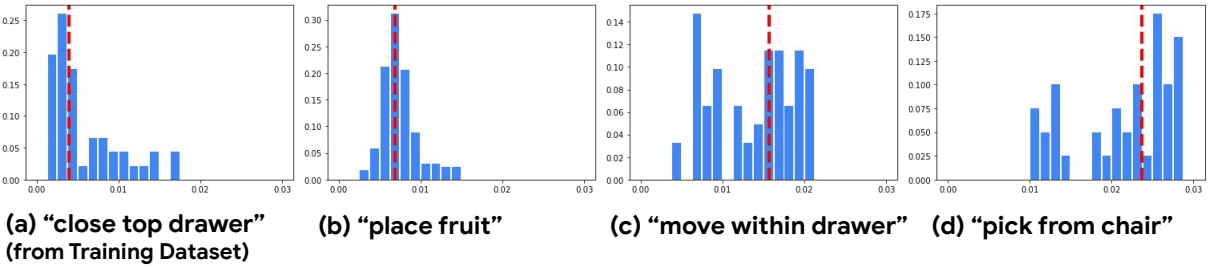


Figure 3.15. We visualize the distribution of Fréchet distances of query trajectories to the most similar training trajectories, as measured by motion similarity. The red line represents the median of the average distance between evaluation trajectories and the most similar training trajectories. Query trajectories of unseen skills in general see larger Fréchet distances to the most similar training trajectories, compared to query trajectories from training skills.

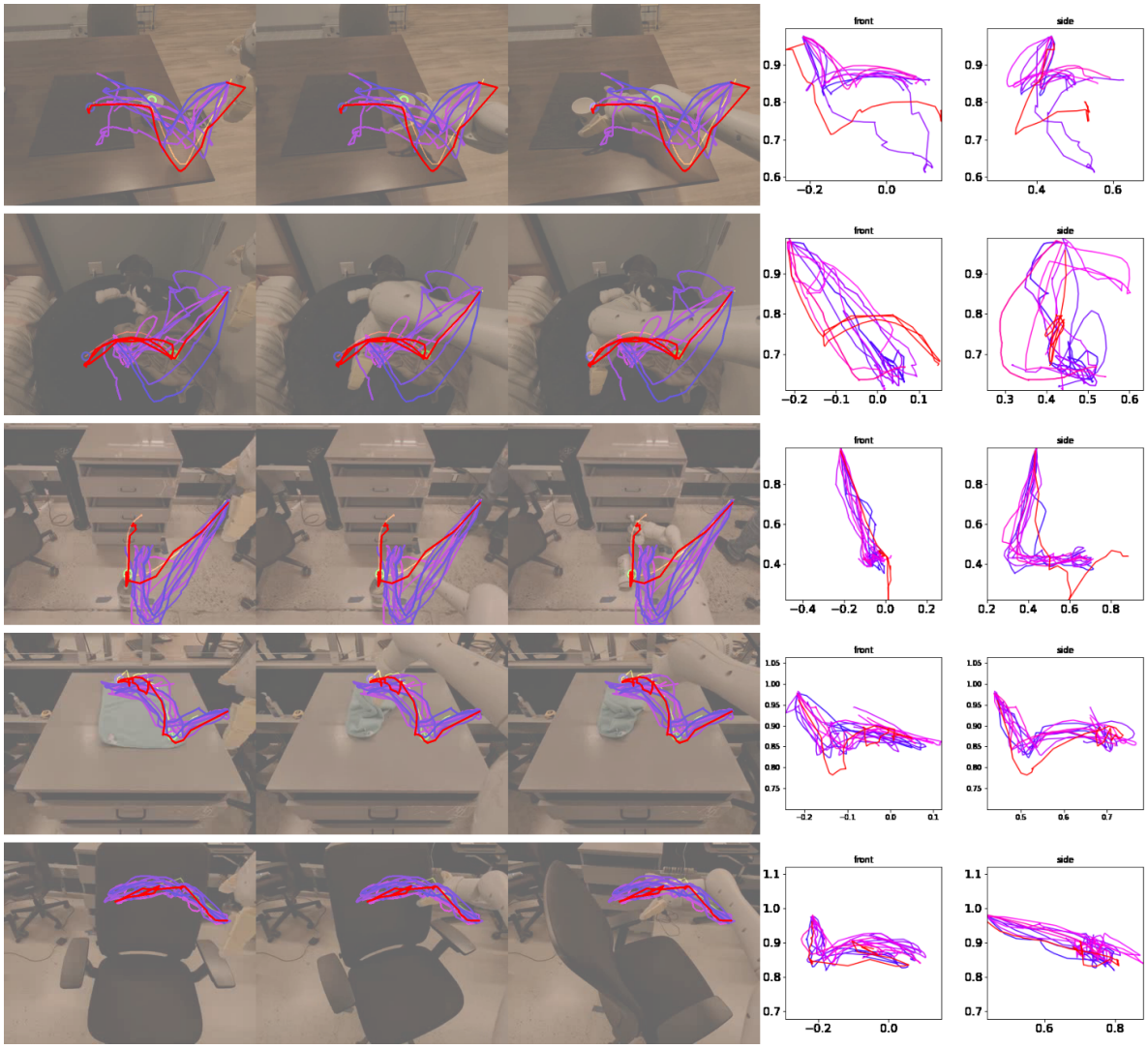


Figure 3.16. Evaluation trajectories for new skills and their 10 closest trajectories from the training set. Each row shows three frames of a skill evaluation rollout, with the executed trajectory and similar training set trajectories overlaid, as well as depicting the trajectories in an orthographic projection in robot base frame looking at the robot from the front and the side. As can be seen, the policy is able to follow the desired trajectories closely and achieve the tasks. While in many cases, in particular in image space, some of the similar trajectories from the training set look very close to the executed trajectory, the front and side view in rows 1 to 4 reveal that the policy at some crucial point has to - and successfully does - deviate from what it has seen during training. E.g., in row 3 the prompt says to go all the way down to pick up a bottle, while all nearest training trajectories are from `close middle drawer`, which doesn't move the gripper down far enough. Additionally, row 5 is an example where for a `swivel chair` prompt trajectory there coincidentally are many very closely matching `move X near Y` training trajectories. However, the prompt here specifies to not close the gripper at the first contact point, which the policy is able to respect.

3.5 Implementation Details for Different Input Modalities

3.5.1 GUI for Human-drawn Trajectory Sketches

As the main trajectory generation method we study is user-specified trajectory drawings, we develop a graphical user interface (GUI) for users to draw trajectory sketches. See Fig. 3.17 for example. Given the current robot camera image, a user can drag and move the mouse to draw curves on the canvas. Then, they can click on the canvas to add markers to indicate gripper closing or opening actions. Additionally, the UI interface also supports simple height annotation. Users can specify the desired height values for pixels they select on the canvas. This height value will be assigned to the closest point on the drawn 2D trajectory. For unannotated points on the 2D trajectory, we interpolate their height values according to annotated ones.

Figure 3.17. Left: The GUI for users to draw trajectory sketches given the robot's current camera image. The 2D trajectory is directly drawn by manual input, which can then be annotated with interaction markers or waypoints corresponding to user-specified heights. Right: The resulting height-aware trajectory sketch generated according to the output of the UI.

3.5.2 Collecting Human-drawn Trajectory Sketches

For each scene, we use a held-out *RT-Trajectory (2.5D)* policy to explore different trajectory "prompts" given a budget of trials, and save the trajectory sketch of the first successful episode. We refer to such process as "prompt engineering" (Sec. 3.4.4). If all attempts fail, we just save the trajectory sketch from the last episode. *RT-Trajectory* policies used for evaluation

are trained with different random seeds and evaluated with the saved trajectory sketches as conditioning. Note that we observe that our evaluated policies can have non-zero success rates on the scenes where we fail to find a successful episode during "prompt engineering".

3.5.3 Human hand pose estimation

We employ Mediapipe [67] to detect the human hand pose represented as 21 landmarks from the 2D image at each video frame. The two landmarks on the thumb and another two landmarks on the index finger are used to represent a parallel gripper. The 2D landmarks are lifted to 3D given the depth map. We then interpolate the end-effector pose from these four points. We manually annotate the key frames at which the hand begins to grasp and release the target object. Given estimated end-effector poses and key frames for interaction, we can generate a trajectory sketch per video.

3.5.4 Implementation Details for *RT-1-Goal*

The network architecture of *RT-1-Goal* is the same as *RT-Trajectory*, except a goal image is used instead of a trajectory sketch. To acquire goal conditioning for training, we use the last observation of each episode as the goal image for all frames in the episode. For the quantitative comparison in Sec. 3.4.2, the image of the last step of the episode (Sec. 3.5.2) used to generate the trajectory sketch for each scene is saved as the goal conditioning for evaluation.

3.6 Additional Visualization

Fig. 3.18 visualizes the example rollouts of unseen skills.

Fig 3.19 shows the rollouts of example evaluations in realistic scenarios mentioned in Sec. 3.4.4. We showcase additional evaluations in Fig. 3.20. Notably, we find that *RT-Trajectory* is quite robust to various simultaneous visual distribution shifts including new buildings, new backgrounds, new distractors, new lighting conditions, new objects, and new furniture textures. In addition, these realistic "in the wild" evaluations were not ran in controlled laboratory

environments, so the evaluations often required generalization to new heights or furniture geometries (different shaped drawers, cabinets, or tables).

Figure 3.18. Example rolllouts of 7 unseen skills. The trajectory sketch overlaid on the initial image is visualized. From top to bottom: `Place Fruit`, `Upright and Move`, `Fold Towel`, `Move within Drawer`, `Restock Drawer`, `Pick from Chair`, `Swivel Chair`.

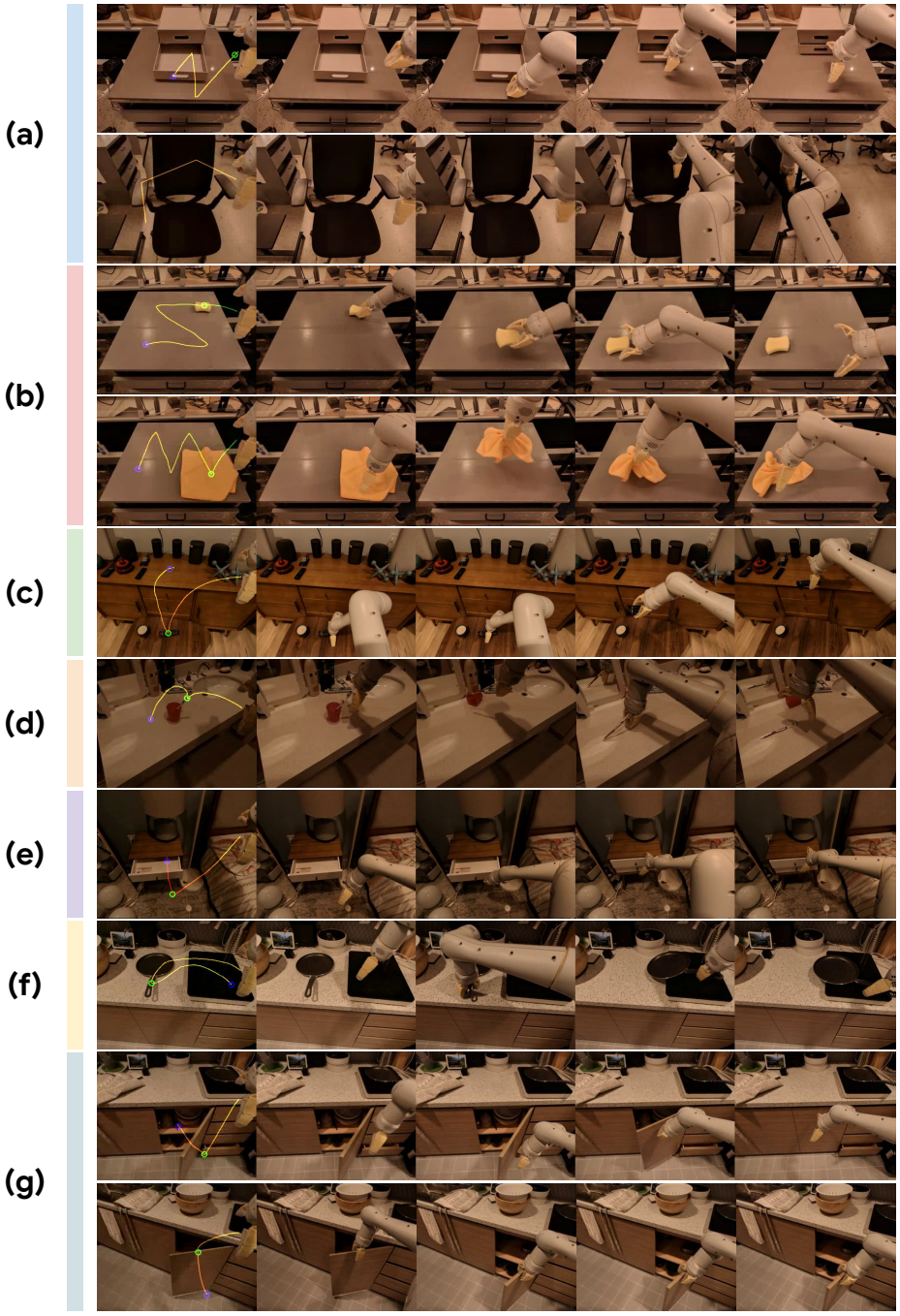


Figure 3.19. Qualitative examples of emergent capabilities of *RT-Trajectory* in realistic scenarios beyond the training settings: (a) new articulated objects requiring novel motion strategies, (b) new circuitous motions requiring multiple turns, (c) new living room setting with a new height, object, and background, (d) new bathroom setting with precise picking from a cup, (e) new bedroom setting with a drawer at a new height, (f) new kitchen setting with an unseen pan requiring placement onto a new stove, and (g) new kitchen setting with a new pivot hinge that requires a new motion for opening and closing.

Figure 3.20. Visualizing additional interesting examples of *RT-Trajectory*'s generalization performance in new scenarios. These include a novel kitchen room setting with an unseen cup and unseen placemat, a new living room room setting with new manipulation objects with new furniture pieces in new heights, and a bathroom setting with harsh lighting and different table height.

3.7 Conclusion and Limitations

In this work, we propose a novel policy-conditioning method for training robot manipulation policies capable of generalizing to tasks and motions that are significantly beyond the training data. Key to our proposed approach is a 2D trajectory sketch representation for specifying manipulation tasks. Our trained trajectory sketch-conditioned policy enjoys controllability from visual trajectory sketch guidance, while retaining the flexibility of learning-based policies in handling ambiguous scenes and generalization to novel semantics. We evaluate our proposed approach on 7 diverse manipulation skills that were never seen during training and benchmark against three baseline methods. Our proposed method achieves a success rate of 67%, significantly outperforming the best prior state-of-the-art methods, which achieved 26%.

Though we demonstrate that our proposed approach achieves encouraging generalization capabilities for novel manipulation tasks, there are a few remaining limitations. First, we currently assume that the robot remains stationary and only uses the end-effector for performing useful manipulation motions. Extending the idea to mobile-manipulation scenarios that allow the robot to manipulate with whole-body control is a promising direction to explore. Second, our trained policy makes its best effort in following the trajectory sketch guidance. However, a user may want to specify spatial regions where the guidance is more strictly enforced, such as when to avoid fragile objects during movement. Thus, an interesting future direction is to enable systems to use trajectory sketches to handle different types of constraints.

Acknowledgement

Chapter 3, in full, is a reprint of the material published in the 2024 International Conference on Learning Representations (ICLR): "RT-Trajectory: Robotic Task Generalization via Hindsight Trajectory Sketches" (Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, Ted Xiao). The dissertation author was the primary investigator and author of this paper.

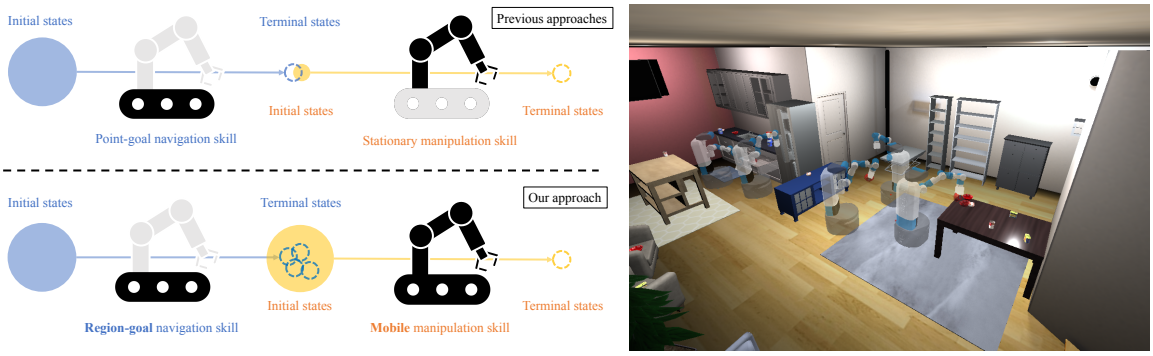
Chapter 4

Improving Skill Formulations for Robust Skill Chaining

We study a modular approach to tackle long-horizon mobile manipulation tasks for object rearrangement, which decomposes a full task into a sequence of subtasks. To tackle the entire task, prior work chains multiple stationary manipulation skills with a point-goal navigation skill, which are learned individually on subtasks. Although more effective than monolithic end-to-end RL policies, this framework suffers from compounding errors in skill chaining, *e.g.*, navigating to a bad location where a stationary manipulation skill can not reach its target to manipulate. To this end, we propose that the manipulation skills should include mobility to have flexibility in interacting with the target object from multiple locations and at the same time the navigation skill could have multiple end points which lead to successful manipulation. We operationalize these ideas by implementing mobile manipulation skills rather than stationary ones and training a navigation skill trained with region goal instead of point goal. We evaluate our multi-skill mobile manipulation method **M3** on 3 challenging long-horizon mobile manipulation tasks in the Home Assistant Benchmark (HAB), and show superior performance as compared to the baselines.

4.1 Introduction

Building AI with embodiment is an important future mission of AI. Object rearrangement [4] is considered as a canonical task for embodied AI. The most challenging rearrangement



(a) Method Overview **(b)** The Home Assistant Benchmark

Figure 4.1. 4.1a provides an overview of our multi-skill mobile manipulation (**M3**) method. The inactive part of the robot is colored gray. Previous approaches exclusively activate either the mobile platform or manipulator for each skill, and suffer from compounding errors in skill chaining given limited composability of skills. We introduce mobility to manipulation skills, which effectively enlarges the feasible initial set, and a region-goal navigation reward to facilitate learning the navigation skill. 4.1b illustrates one task (*SetTable*) in the Home Assistant Benchmark [109], where the robot needs to navigate in the room, open the drawers or fridge, pick multiple objects in drawers or fridge and place them on the table. Best viewed in motion at the project website.

tasks [109, 24, 31] are often long-horizon mobile manipulation tasks, which demand both navigation and manipulation abilities, *e.g.*, to move to certain locations and to pick or place objects. It is challenging to learn a monolithic RL policy for complex long-horizon mobile manipulation tasks, due to challenges such as high sample complexity, complicated reward design, and inefficient exploration. A practical solution to tackle a long-horizon task is to decompose it into a set of subtasks, which are tractable, short-horizon, and compact in state or action spaces. Each subtask can be solved by designing or learning a skill, so that a sequence of skills can be chained to complete the entire task [61, 19, 62, 60]. For example, skills for object rearrangement can be picking or placing objects, opening or closing fridges and drawers, moving chairs, navigating in the room, *etc*.

Achieving successful object rearrangement using this modular framework requires careful subtask formulation such that skills trained for these subtasks can be chained together effectively. We define three desirable properties for skills to solve diverse long-horizon tasks: **achievability,**

composability, and reusability. Note that we assume each subtask is associated with a set of initial states. Then, *achievability* quantifies the portion of initial states solvable by a skill. A pair of skills are *composable* if the initial states achievable by the succeeding skill can encompass the terminal states of the preceding skill. This encompassment requirement is necessary to ensure robustness to mild compounding errors. However, trivially enlarging the initial set of a subtask increases learning difficulty and may lead to many unachievable initial states for the designed/learned skill. Last, a skill is *reusable* if it can be directly chained without or with limited fine-tuning [19, 60]. According to our experiments, effective subtask formulation is critical though largely overlooked in the literature.

In the context of mobile manipulation, skill chaining poses many challenges for subtask formulation. For example, an imperfect navigation skill might terminate at a bad location where the target object is out of reach for a stationary manipulation skill [109]. To tackle such "hand-off" problems, we investigate how to formulate subtasks for mobile manipulation. First, we replace stationary (fixed-base) manipulation skills with mobile counterparts, which allow the base to move when the manipulation is undertaken. We observe that mobile manipulation skills are more robust to compounding errors in skill chaining, and enable the robot to make full use of its embodiment to better accomplish subtasks, *e.g.*, finding a better location with less clutter and fewer obstacles to pick an object. We emphasize how to generate initial states of manipulation skills as a trade-off between *composability* and *achievability* in Sec 4.4.1. Second, we study how to translate the start of manipulation skills to the navigation reward, which is used to train the navigation skill to connect manipulation skills. Note that the goal position in mobile manipulation plays a very different role from that in point-goal [122, 54] navigation. On the one hand, the position of a target object (*e.g.*, on the table or in the fridge) is often not directly navigable; on the other hand, a navigable position close to the goal position can be infeasible due to kinematic and collision constraints. Besides, there exist multiple feasible starting positions for manipulation skills, yet previous works such as [109] train the navigation skill to learn a single one, which is selected heuristically and may not be suitable for stationary manipulation. Thanks to the

flexibility of our mobile manipulation skills, we devise a region-goal navigation reward to address those issues, detailed in Sec 4.4.2.

In this work, we present our improved multi-skill mobile manipulation method **M3**, where mobile manipulation skills are chained by the navigation skill trained with our region-goal navigation reward. It achieves an average success rate of 63% on 3 long-horizon mobile manipulation tasks in the Home Assistant Benchmark [109], as compared to 50% for our best baseline. Fig 4.1 provides an overview of our method and tasks. Our contributions are listed as follows:

1. We study how to formulate mobile manipulation skills, and empirically show that they are more robust to compounding errors in skill chaining than stationary counterparts;
2. We devise a region-goal navigation reward for mobile manipulation, which shows better performance and stronger generalizability than the point-goal counterpart in previous works;
3. We show that our improved multi-skill mobile manipulation pipeline can achieve superior performance on long-horizon mobile manipulation tasks without bells and whistles, which can serve as a strong baseline for future study.

4.2 Related Work

4.2.1 Mobile Manipulation

Rearrangement [4] is "to bring a given physical environment into a specified state". We refer readers to [4] for a comprehensive survey. Many existing RL tasks can be considered as instances of rearrangement, *e.g.*, picking and placing rigid objects [140, 133] or manipulating articulated objects [114, 82]. However, they mainly focus on stationary manipulation [114, 140, 133] or individual, short-horizon skills [82]. Recently, several benchmarks like Home Assistant Benchmark (HAB) [109], ManipulaTHOR [24] and ThreeDWorld Transport Challenge [31], are proposed to study long-horizon mobile manipulation tasks. They usually demand that the robot

rearranges household objects in a room, requiring exploration and navigation [3, 16] between interacting with objects entirely based on onboard sensing, without any privileged state or map information.

Mobile manipulation [97] refers to "robotic tasks that require a synergistic combination of navigation and interaction with the environment". It has been studied long in the robotics community. [84] provides a summary of traditional methods, which usually require perfect knowledge of the environment. One example is task-and-motion-planning (TAMP) [105, 32, 33]. TAMP relies on well-designed state proposals (grasp poses, robot positions, *etc*.) to sample feasible trajectories, which is computationally inefficient and unscalable for complicated scenarios.

Learning-based approaches enable the robot to act according to visual observations. [126] proposes a hierarchical method for mobile manipulation in iGibson [127], which predicts either a high-level base or arm action by RL policies and executes plans generated by motion-planning to achieve the action. However, the arm action space is specially designed for a primitive action *pushing*. [107] develops a real-world RL framework to collect trash on the floor, with separate navigation and grasping policies. [24, 84] train an end-to-end RL policy to tackle mobile pick-and-place in ManipulaTHOR [24]. However, the reward function used to train such an end-to-end policy usually demands careful tuning. For example, [84] shows that a minor modification (a penalty for disturbance avoidance) can lead to a considerable performance drop. The vulnerability of end-to-end RL approaches restricts scalability. Most prior works in both RL and robotics separate mobile the platform and manipulator, to "reduce the difficulty to solve the inverse kinematics problem of a kinematically redundant system" [101, 99]. [116] trains an end-to-end RL policy based on the object pose and proprioception to simultaneously control the base and arm. It focuses on picking a single object up in simple scenes, while our work addresses long-horizon rearrangement tasks that require multiple skills.

[109] adopts a different hierarchical approach for mobile manipulation. It uses task-planning [28] to generate high-level symbolic goals, and individual skills are trained by RL to accomplish those goals. It outperforms the monolithic end-to-end RL policy and the classical

sense-plan-act robotic pipeline. It is scalable since skills can be composited to solve different tasks, and benefit from progress in individual skill learning [133, 82]. Moreover, different from other benchmarks, the HAB features continuous motor control (base and arm), interaction with articulated objects (opening drawers and fridges), and complicated scene layouts. Thus, we choose the HAB as the platform to study long-horizon mobile manipulation.

4.2.2 Skill Chaining for Long-horizon Tasks

[109] observes that sequentially chaining multiple skills suffers from "hand-off" problems, where a preceding skill terminates at a state that the succeeding skill has either never seen during training or is infeasible to solve. [61] proposes to learn a transition policy to connect primitive skills, but assumes that such a policy can be found through random exploration. [60] regularizes the terminal state distribution of a skill to be close to the initial set of the following skill, through a reward learned with adversarial training. Most prior skill chaining methods focus on fine-tuning learned skills. In this work, we instead focus on subtask formulation for skill chaining, which directly improves composability and reusability without additional computation.

4.3 Preliminary

4.3.1 Home Assistant Benchmark (HAB)

The Home Assistant Benchmark (HAB) [109] includes 3 long-horizon mobile manipulation rearrangement tasks (*TidyHouse, PrepareGroceries, SetTable*) based on the ReplicaCAD dataset, which contains a rich set of 105 indoor scene layouts. For each episode (instance of task), rigid objects from the YCB [14] dataset are randomly placed on annotated supporting surfaces of receptacles, to generate clutter in a randomly selected scene. Here we provide a brief description of these tasks.

TidyHouse: Move 5 objects from starting positions to goal positions. Objects and goals are located in open receptacles (*e.g.*, table, kitchen counter) rather than containers. Complex

scene layouts, diverse receptacles, dense clutter all pose challenges. The task implicitly favors collision-free behavior since a latter target object might be knocked out of reach when a former object is moved by the robot.

PrepareGroceries: Move 2 objects from the fridge to the counters and move an object from the counter to the fridge. The fridge is fully open initially. The task requires picking and placing an object in a cluttered receptacle with restricted space.

SetTable: Move a bowl from a drawer to a table, and move a fruit from the fridge to the bowl on the table. Both the drawer and fridge are closed initially. The task requires interaction with articulated objects as well as picking objects from containers.

All the tasks demand onboard sensing instead of privileged information (*e.g.*, ground-truth object positions and navigation map). All the tasks use the GeometricGoal [4] specification (s_0, s_*), which describes the initial 3D (center-of-mass) position s_0 of the target object and the goal position s_*. For example, *TidyHouse* is specified by 5 tuples $\{(s_0^i, s_*^i)\}_{i=1...5}$.

4.3.2 Subtask and Skill

In this section, we present the definition of subtask and skill in the context of reinforcement learning. A long-horizon task can be formulated as a Markov decision process (MDP) [1] defined by a tuple $(\mathcal{S}, \mathcal{A}, R, P, \mathcal{I})$ of state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R(s, a, s')$, transition distribution $P(s'|s, a)$, initial state distribution $\mathcal{I}$. A subtask ω is a smaller MDP $(\mathcal{S}, \mathcal{A}_\omega, R_\omega, P, \mathcal{I}_\omega)$ derived from the original MDP of the full task. A skill (or policy), which maps a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$, is learned for each subtask by RL algorithms.

[109] introduces several parameterized skills for the HAB: *Pick, Place, Open fridge, Close fridge, Open drawer, Close drawer, Navigate*. Each skill takes a 3D position as input, either s_0 or s_*. See Sec. 4.6.2 for more details. Here, we provide a brief description of these skills:

- **Pick**(s_0): pick the object initialized at s_0
- **Place**(s_*): place the held object at s_*

[1] To be precise, the tasks studied in this work are partially observable Markov decision process (POMDP).

- **Open [container]**(s): open the container containing the object initialized at s or the goal position s
- **Close [container]**(s): close the container containing the object initialized at s or the goal position s
- **Navigate**(s): navigate to the start of other skills specified by s

Note that s_0 is constant per episode instead of a tracked object position. Hence, the target object may not be located at s_0 at the beginning of a skill, *e.g.*, picking an object from an opened drawer. Next, we will illustrate how these skills are chained in the HAB.

4.3.3 Skill Chaining

Given a task decomposition, a hierarchical approach also needs to generate high-level actions to select a subtask and perform the corresponding skill. Task planning [28] can be applied to find a sequence of subtasks before execution, with perfect knowledge of the environment. An alternative is to learn high-level actions through hierarchical RL. In this work, we use the subtask sequences generated by a perfect task planner [109]. Here we list these sequences, to highlight the difficulty of tasks [2]:

- **TidyHouse**(s_0^i, s_*^i): $\text{Navigate}(s_0^i) \rightarrow \text{Pick}(s_0^i) \rightarrow \text{Navigate}(s_*^i) \rightarrow \text{Place}(s_*^i)$
- **PrepareGroceries**($s_0^1, s_*^1, s_0^2, s_*^2, s_0^3, s_*^3$): $\text{Navigate}_{\text{fr}}(s_0^1) \rightarrow \text{Pick}_{\text{fr}}(s_0^1) \rightarrow \text{Navigate}(s_*^1)$
  $\rightarrow \text{Place}(s_*^1) \rightarrow \text{Navigate}_{\text{fr}}(s_0^2) \rightarrow \text{Pick}_{\text{fr}}(s_0^2) \rightarrow \text{Navigate}(s_*^2) \rightarrow \text{Place}(s_*^2)$
  $\rightarrow \text{Navigate}(s_0^3) \rightarrow \text{Pick}(s_0^3) \rightarrow \text{Navigate}_{\text{fr}}(s_*^3) \rightarrow \text{Place}_{\text{fr}}(s_*^3)$
- **SetTable**($s_0^1, s_*^1, s_0^2, s_*^2$): $\text{Navigate}_{\text{dr}}(s_0^1) \rightarrow \text{Open}_{\text{dr}}(s_0^1) \rightarrow \text{Pick}_{\text{dr}}(s_0^1)$
  $\rightarrow \text{Navigate}(s_*^1) \rightarrow \text{Place}(s_*^1) \rightarrow \text{Navigate}_{\text{dr}}(s_0^1) \rightarrow \text{Close}_{\text{dr}}(s_0^1)$
  $\rightarrow \text{Navigate}_{\text{fr}}(s_0^2) \rightarrow \text{Open}_{\text{fr}}(s_0^2) \rightarrow \text{Navigate}_{\text{fr}}(s_0^2) \rightarrow \text{Pick}_{\text{fr}}(s_0^2)$
  $\rightarrow \text{Navigate}(s_*^2) \rightarrow \text{Place}(s_*^2) \rightarrow \text{Navigate}_{\text{fr}}(s_0^2) \rightarrow \text{Close}_{\text{fr}}(s_0^2)$

[2] We only list the subtask sequence of *TidyHouse* for one object here for illustration. The containers are denoted with subscripts fr (fridge) and dr (drawer) if included in the skill.

4.4 Subtask Formulation and Skill Learning for Mobile Manipulation

Following the proposed principles (*composability*, *achievability*, *reusability*), we revisit and reformulate subtasks defined in the Home Assistant Benchmark (HAB). The core idea is to enlarge the initial states of manipulation skills to encompass the terminal states of the navigation skill, given our observation that the navigation skill is usually more robust to initial states. However, manipulation skills (*Pick*, *Place*, *Open drawer*, *Close drawer*) in [109], are stationary. The *composability* of a stationary manipulation skill is restricted, since its feasible initial states are limited due to kinematic constraints. For instance, the robot can not open the drawer if it is too close or too far from the drawer. Therefore, these initial states need to be carefully designed given the trade-off between *composability* and *achievability*, which is not scalable and flexible. On the other hand, the navigation skill, which is learned to navigate to the start of manipulation skills, is also restricted by stationary constraints, since it is required to precisely terminate at a small set of "good" locations for manipulation. To this end, we propose to replace stationary manipulation skills with mobile counterparts. Thanks to mobility, mobile manipulation skills can have better *composability* without sacrificing much *achievability*. For example, a mobile manipulator can learn to first get closer to the target and then manipulate, to compensate for errors from navigation. It indicates that the initial states can be designed in a more flexible way, which also enables us to design a better navigation reward to facilitate learning.

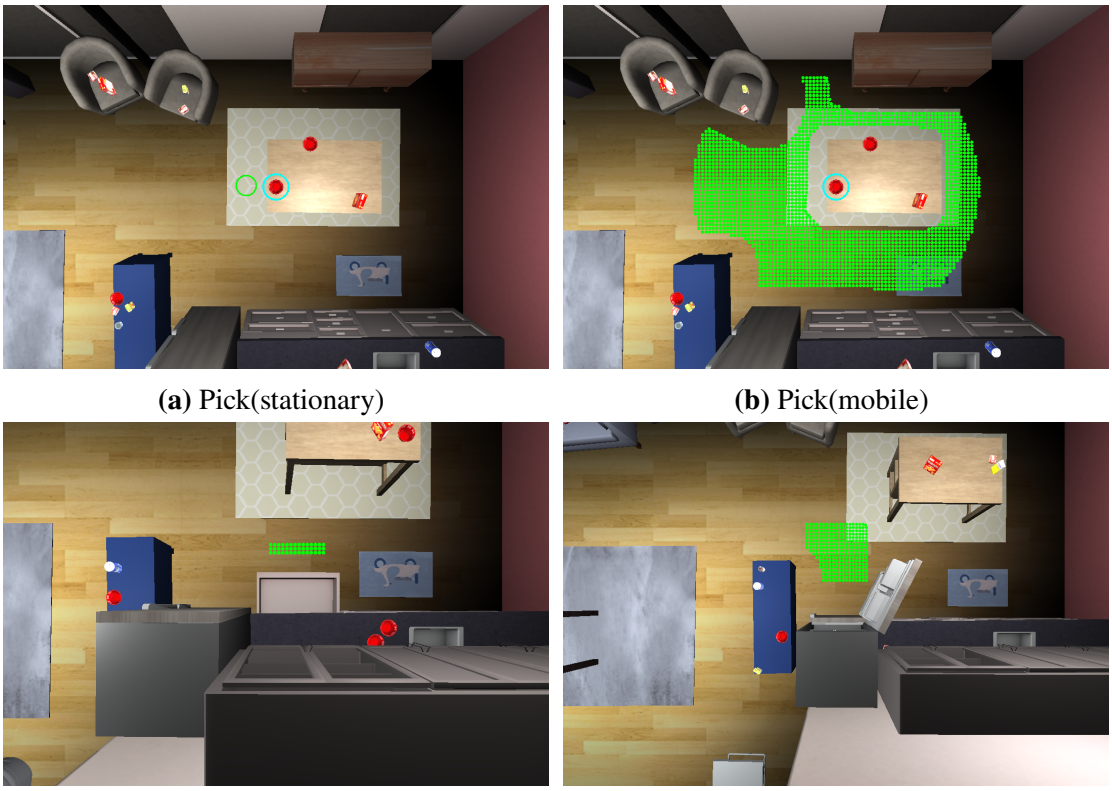
In the context of mobile manipulation, the initial state of a skill consists of the robot base position, base orientation, and joint positions. For simplicity, we do not discuss the initial states of rigid and articulated objects in the scene, which are usually defined in episode generation. Moreover, we follow previous works [109, 60] to initialize the arm at its resting position and reset it after each skill in skill chaining. Such a reset operation is common in robotics [33]. Each skill is learned to reset the arm after accomplishing the subtask as in [109]. Furthermore, for base orientation, we follow the heuristic in [109] to make the robot face the target position s_0 or s_*.

4.4.1 Manipulation Skills with Mobility

We first present how initial base positions are generated in previous works. For stationary manipulation, a feasible base position needs to satisfy several constraints, *e.g.*, kinematic (the target is reachable) and collision-free constraints. [109] uses heuristics to determine base positions. For *Pick, Place* without containers (fridge and drawer), a navigable position closest to the target position is selected. For *Pick, Place* with containers, a fixed position relative to the container is selected. For *Open, Close*, a navigable position is randomly selected from a handcrafted region relative to each container. Noise is added to base position and orientation in addition, and infeasible initial states are rejected by constraints. See Fig 4.2 for examples.

The above example indicates the difficulty and complexity to design feasible initial states for stationary manipulation. One naive solution is to enlarge the initial set with infeasible states, but this can hurt learning as shown later in Sec 4.5.4. Besides, rejection sampling can be quite inefficient in this case, and [109] actually computes a fixed number of feasible initial states offline.

Manipulation Skills with Mobility. To this end, we propose to use mobile manipulation skills instead. The original action space (only arm actions) is augmented with base actions. We devise a unified and efficient pipeline to generate initial base positions. Concretely, we first discretize the floor map with a resolution of $5 \times 5cm^2$, and get all navigable (grid) positions. Then, different candidates are computed from these positions based on subtasks. Candidates are either within a radius (*e.g.*, 2m) around the target position for *Pick, Place*, or a region relative to the container for *Open, Close*. Finally, a feasible position is sampled from the candidates with rejection and noise. Compared to stationary manipulation, the rejection rate of our pipeline is much lower, and thus can be efficiently employed on-the-fly during training. See Fig 4.2 for examples.



(a) Pick(stationary) **(b)** Pick(mobile)

(c) Close drawer **(d)** Close fridge

Figure 4.2. Initial base positions of manipulation skills. We only show the examples for *Pick, Close drawer, Close fridge*, as *Place, Open drawer, Open fridge* share the same initial base positions respectively. Positions are visualized as green points on the floor. The target object in *Pick* is highlighted by a circle in cyan. Note that the initial base position of *Pick(stationary)* is a single navigable position closest to the object.

4.4.2 Navigation Skill with Region-Goal Navigation Reward

The navigation skill is learned to connect different manipulation skills. Hence, it needs to terminate within the set of initial achievable states of manipulation skills. We follow [109] to randomly sample a navigable base position and orientation as the initial state of navigation skill. The challenge is how to formulate the reward function, which implicitly defines desirable terminal states. A common navigation reward [122] is the negative change of geodesic distance to a single 2D goal position on the floor. [109] extends it for mobile manipulation, which introduces the negative change of angular distance to the desired orientation (facing the target). The resulting

reward function, $r_t(s, a)$, for state s and action a is the following (Eq 4.1):

$$r_t(s,a) = -\Delta_{geo}(g) - \lambda_{ang}\Delta_{ang} I_{[d_t^{geo}(g)\leq \tilde{D}]} + \lambda_{succ} I_{[d_t^{geo}(g)\leq D \wedge d_t^{ang}\leq \Theta]} - r_{slack} \tag{4.1}$$

$\Delta_{geo}(g) = d_t^{geo}(x_t^{base}, g) - d_{t-1}^{geo}(x_{t-1}^{base}, g)$, where $d_t^{geo}(x_t^{base}, g)$ is the geodesic distance between the current base position x_t^{base} and the 2D goal position g. $d_t^{geo}(g)$ is short for $d_t^{geo}(x_t^{base}, g)$. $\Delta_{ang} = d_t^{ang} - d_{t-1}^{ang} = \|\theta_t - \theta^*\|_1 - \|\theta_{t-1} - \theta^*\|_1$, where θ_t is the current base orientation, and θ^* is the target orientation. Note that the 2D goal on the floor is different from the 3D goal specification for manipulation subtasks. $I_{[d_t^{geo}\leq \tilde{D}]}$ is an indicator of whether the agent is close enough to the 2D goal, where $\tilde{D}$ is a threshold. $I_{[d_t^{geo}\leq D \wedge d_t^{ang}\leq \Theta]}$ is an indicator of navigation success, where D and Θ are thresholds for geodesic and angular distances. r_{slack} is a slack penalty. $\lambda_{ang}, \lambda_{succ}$ are hyper-parameters.

This reward has several drawbacks: 1) A single 2D goal needs to be assigned, which should be an initial base position of manipulation skills. It is usually sampled with rejection, as explained in Sec 4.4.1. It ignores the existence of multiple reasonable goals, introduces ambiguity to the reward (hindering training), and leads the skill to memorize (hurting generalization). 2) There is a hyperparameter $\tilde{D}$, which defines the region where the angular term Δ_{ang} is considered. However, it can lead the agent to learn the undesirable behavior of entering the region with a large angular distance, *e.g.*, backing onto the target.

Region-Goal Navigation Reward. To this end, we propose a region-goal navigation reward for training the navigation skill. Inspired by object-goal navigation, we use the geodesic distance [3] between the robot and a region of 2D goals on the floor instead of a single goal. Thanks to the flexibility of our mobile manipulation skills, we can simply reuse the candidates (Sec 4.4.1) for their initial base positions as the navigation goals. However, these candidates are not all collision-free. Thus, we add a collision penalty $r_{col} = \lambda_{col} C_t$ to the reward, where C_t is

[3]The geodesic distance to a region can be approximated by the minimum of all the geodesic distances to grid positions within the region.

the current collision force and λ_{col} is a weight. Besides, we simply remove the angular term, and find that the success reward is sufficient to encourage correct orientation. Our region-goal navigation reward is as follows:

$$r_t(s,a) = -\Delta_{geo}(\{g\}) + \lambda_{succ} I_{[d_t^{geo}(\{g\}) \leq D \wedge d_t^{ang} \leq \Theta]} - r_{col} - r_{slack} \tag{4.2}$$

4.5 Experiments

4.5.1 Experimental Setup

We use the ReplicaCAD dataset and the Habitat 2.0 simulator [109] for our experiments. The ReplicaCAD dataset contains 5 macro variations, with 21 micro variations per macro variation [4]. We hold out 1 macro variation to evaluate the generalization of unseen layouts. For the rest of the 4 macro variations, we split 84 scenes into 64 scenes for training and 20 scenes to evaluate the generalization of unseen configurations (object and goal positions). For each task, we generate 6400 episodes (64 scenes) for training, 100 episodes (20 scenes) to evaluate cross-configuration generalization, and another 100 episodes (the hold-out macro variation) to evaluate cross-layout generalization. The robot is a Fetch [98] mobile manipulator with a 7-DoF arm and a parallel-jaw gripper. See Sec. 4.6.1 for more details about the setup and dataset generation.

Observation space: The observation space includes head and arm depth images (128×128), arm joint positions (7-dim), end-effector position (3-dim) in the base frame, goal positions (3-dim) in both base and end-effector frames, as well as a scalar to indicate whether an object is held. The goal position, depending on subtasks, can be either the initial or desired position of the target object. We assume a perfect GPS+Compass sensor and proprioceptive sensors as in [109], which are used to compute the relative goal positions. For the navigation skill, only the head depth image and the goal position in the base frame are used.

[4] Each macro variation has a different, semantically plausible layout of large furniture (*e.g.*, kitchen counter and fridge) while each micro variation is generated through perturbing small furniture (*e.g.*, chairs and tables).

Action space: The action space is a 10-dim continuous space, including 2-dim base action (linear forwarding and angular velocities), 7-dim arm action, and 1-dim gripper action. Grasping is abstract as in [4, 109, 24]. If the gripper action is positive, the object closest to the end-effector within 15cm will be snapped to the gripper; if negative, the gripper will release any object held. For the navigation skill, we use a discrete action space, including a stop action, as in [131, 109]. A discrete action will be converted to continuous velocities to move the robot, while arm and gripper actions are masked out.

Hyper-parameters: We train each skill by the PPO [100] algorithm. The visual observations are encoded by a 3-layer CNN as in [109]. The visual features are concatenated with state observations and previous action, followed by a 1-layer GRU and linear layers to output action and value. Each skill is trained with 3 different seeds. See Sec. 4.6.3 for details.

Metrics: Each HAB task consists of a sequence of subtasks to accomplish, as illustrated in Sec 4.3.3. The completion of a subtask is conditioned on the completion of its preceding subtask. We report progressive completion rates of subtasks, and the completion rate of the last subtask is thus the success rate of the full task. For each evaluation episode, the robot is initialized at a random base position and orientation without collision, and its arm is initialized at the resting position. The completion rate is averaged over 9 different runs [5].

4.5.2 Baselines

We denote our method by **M3**, short for a multi-skill mobile manipulation pipeline where mobile manipulation skills (**M**) are chained by the navigation skill trained with our region-goal navigation reward (**R**). We compare our method with several RL baselines. All baselines follow the same experimental setup in Sec 4.5.1 unless specified. We refer readers to [109] for a sense-plan-act baseline, which is shown to be inferior to the skill chaining pipeline emphasized in this work. Stationary manipulation skills and point-goal navigation reward are denoted by **S** and **P**.

[5] 3 seeds for RL training multiplied by 3 seeds for initial states

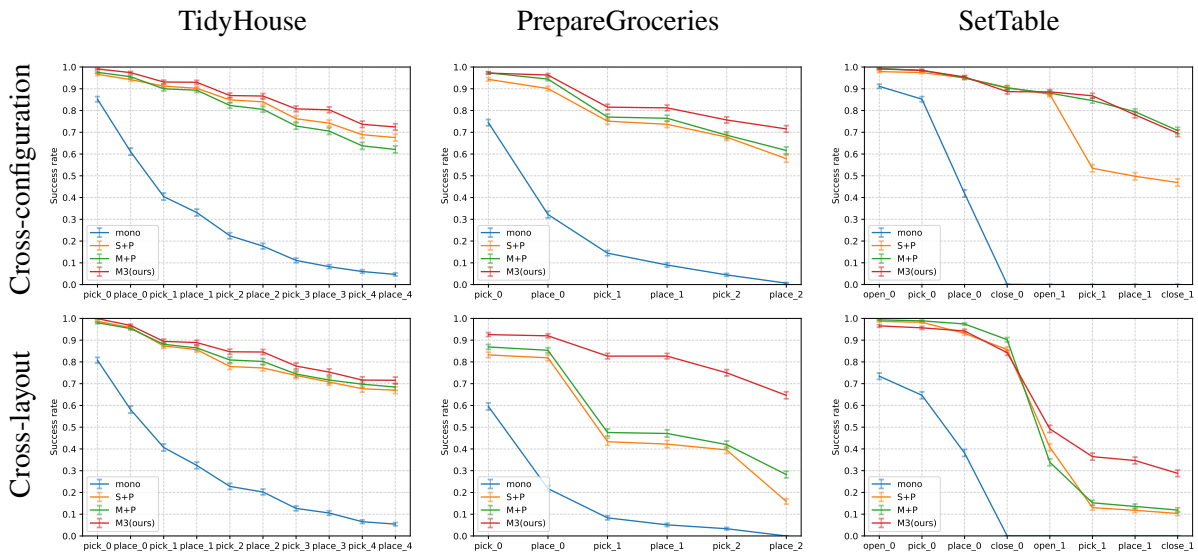


Figure 4.3. Progressive completion rates for HAB [109] tasks. The x-axis represents progressive subtasks. The y-axis represents the completion rate of each subtask. The mean and standard error for 100 episodes over 9 seeds are reported. Best viewed zoomed.

Monolithic RL (mono): This baseline is an end-to-end RL policy trained with a combination of reward functions of individual skills. See Sec. 4.6.5 for more details.

Stationary manipulation skills + point-goal navigation reward (S+P): This baseline is TaskPlanning+SkillsRL (TP+SRL) introduced in [109], where stationary manipulation skills are chained by the navigation skill trained with the point-goal navigation reward. Compared to the original implementation, we make several improvements, including better reward functions and training schemes. For reference, the original success rates of all HAB tasks are nearly zero.

Mobile manipulation skills + point-goal navigation reward (M+P): Compared to our **M3**, this baseline does not use the region-goal navigation reward. It demonstrates the effectiveness of proposed mobile manipulation skills. Note that the point-goal navigation reward is designed for the start of stationary manipulation skills.

4.5.3 Results

Fig 4.3 shows the progressive completion rates of different methods on all tasks. Our method **M3** achieves an average success rate of 71.2% in the cross-configuration setting, and

55.0% in the cross-layout setting, over all 3 tasks. It outperforms all the baselines in both settings, namely **mono** (1.8%/1.8%), **S+P** (57.4%/31.1%) and **M+P** (64.9%/36.2%). First, all the modular approaches show much better performance than the monolithic baseline, which verifies the effectiveness of modular approaches for long-horizon mobile manipulation tasks. Mobile manipulation skills are in general superior to stationary ones (**M+P** *vs*.**S+P**). Fig 4.4 provides an example where mobile manipulation skills can compensate for imperfect navigation. Furthermore, our region-goal navigation reward can reduce the ambiguity of navigation goals to facilitate training (see training curves in Sec. 4.6.2). Since it does not require the policy to memorize ambiguous goals, the induced skill shows better generalizability, especially in the cross-layout setting (55.0% for **M3** *vs*.36.2% for **M+P**).

We present more quantitative metrics in addition to progressive completion rates for the main experiments on 3 HAB tasks. We report the number of successfully placed objects and the average distance between objects and goals in Table 4.1 and 4.2. These metrics are analogous to `%FIXEDSTRICT` and `%E` in [119].

Table 4.1. The number of successfully placed objects for HAB tasks. The metrics in the cross-configuration/cross-layout setting are reported. The number of objects to place is shown along with the name of each task.

Method	TidyHouse (5)	PrepareGroceries (3)	SetTable (2)
S+P	4.58/4.56	2.33/1.54	1.45/1.03
M+P	4.45/4.54	2.45/1.79	1.71/1.08
M3 (ours)	**4.64/4.60**	**2.56/2.44**	**1.71/1.22**

Table 4.2. Average distance between objects and goals for HAB tasks. The metrics in the cross-configuration/cross-layout setting are reported. Note that the average distance is sensitive to outliers.

Method	TidyHouse	PrepareGroceries	SetTable
S+P	0.245/0.256	0.338/0.982	**3.506**/6.481
M+P	1.296/0.821	0.302/0.689	4.165/**4.641**
M3 (ours)	**0.198/0.174**	**0.239/0.337**	5.208/6.345

(a) Stationary Manipulation (S+P)

(b) Mobile Manipulation (M+P)

Figure 4.4. Qualitative comparison between stationary and mobile manipulation. In this example, the point-goal navigation skill terminates between two drawers (1st image). Mobile manipulation manages to open the correct drawer containing the bowl (last image in the bottom row) while stationary manipulation gets confused and finally opens the wrong drawer (last image in the top row). More qualitative results can be found in Sec. 4.8 and on our project website.

4.5.4 Ablation Studies

We conduct several ablation studies to show that mobile manipulation skills are more flexible to formulate than stationary ones, and to understand the advantage of our navigation reward.

Can initial states be trivially enlarged? We conduct experiments to understand to what extent we can enlarge the initial states of manipulation skills given the trade-off between *achievability* and *composability*. In the **S(L)+P** experiment, we simply replace the initial states of stationary manipulation skills with those of mobile ones. The success rates of stationary manipulation skills on subtasks drop by a large margin, *e.g.*, from 95% to 45% for *Pick* on *TidyHouse*. Fig 4.5 shows that **S(L)+P** (37.7%/18.1%) is inferior to both **S+P** (57.4%/31.1%) and **M+P** (64.9%/36.2%). It indicates that stationary manipulation skills have a much smaller set of feasible initial states compared to mobile ones, and including infeasible initial states during training can hurt performance significantly.

Is the collision penalty important for the navigation skill? Our region-goal navigation

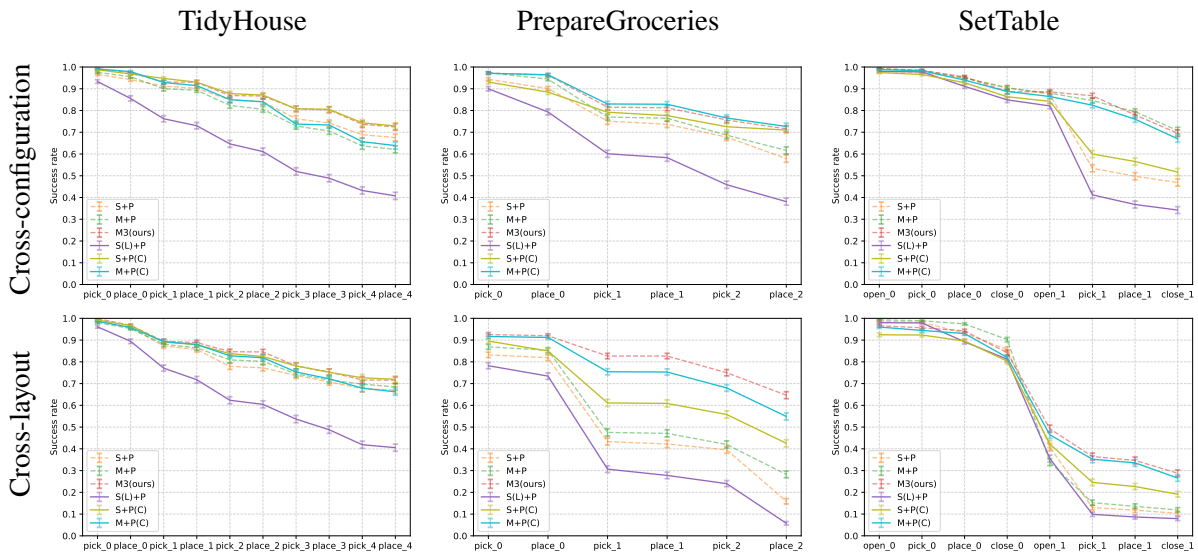


Figure 4.5. Progressive completion rates for HAB tasks. The x-axis represents progressive subtasks. The y-axis represents the completion rate of each subtask. Results of ablation experiments are presented with solid lines. The mean and standard error for 100 episodes over 9 seeds are reported.

reward benefits from unambiguous region goals and the collision penalty. We add the collision penalty to the point-goal navigation reward (Eq 4.1) in **S+P(C)** and **M+P(C)** experiments. Fig 4.5 shows that the collision penalty significantly improves the success rate: **S+P(C)** (65.2%/44.6%) *vs*.**S+P** (57.4%/31.1%) and **M+P(C)** (67.9%/49.2%) *vs*.**M+P** (64.9%/36.2%). A collision-aware navigation skill can avoid disturbing the environment, *e.g.*, accidentally closing the fridge before placing an object in it. Besides, **M+P(C)** is still inferior to our **M3** (71.2%/55.0%). It implies that reducing the ambiguity of navigation goals helps learn more robust and generalizable navigation skills.

Impact of different initial state distributions

We study the impact of different initial state distributions on mobile manipulation skills. We enlarge initial states by changing the distributions of the initial base position (the radius around the target) and orientation. For reference, the maximum radius around the target is set to 2m in the main experiments (Sec 4.5). Several experiments are conducted: **M(S)+R**, **M(L1)+R**, **M(L2)+R**, **M(L3)+R**. **M(S)+R**, **M(L1)+R** and **M(L2)+R** stand for the experiments where the

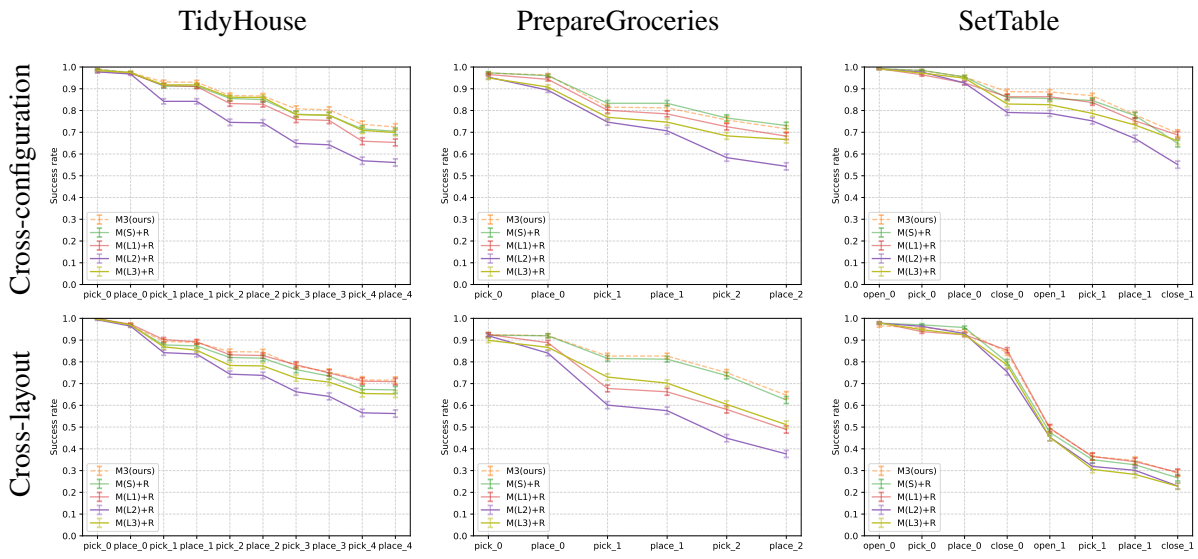


Figure 4.6. Progressive completion rates for HAB [109] tasks. The x-axis represents progressive subtasks. The y-axis represents the completion rate of each subtask. Results of ablation experiments are presented with solid lines. The mean and standard error for 100 episodes over 9 seeds are reported.

maximum radii around the target are set to 1.5m, 2.5m and 4m respectively. **M(L3)+R** keeps the radius as 2m, but samples the initial base orientation from $[-\pi, \pi]$, instead of using the direction facing towards the target. Fig 4.6 shows the quantitative results. Enlarging the initial states in general leads to performance degradation. Compared to **M3** (71.2%/55.0%), **M(L1)+R** (67.4%/49.7%) and **M(L3)+R** (67.5%/46.4%) show moderate performance drop. **M(L2)+R** (55.2%/38.9%) shows the largest performance drop, which indicates that mobile manipulation skills are not able to handle long-range navigation yet. Moreover, **M(S)+R** (69.5%/52.1%) performs on par with **M3**. It implies that there usually exists a "sweet spot" of the initial state distribution for mobile manipulation skills as a trade-off between *achievability* and *composability*.

Besides, we extend the S(L)+P experiment described in Sec 4.5.4, where we simply replace the initial states of stationary manipulation skills with those of mobile ones. We reject the initial states that the target is not reachable due to the kinematic constraint. The constraint is checked via inverse kinematics (IK). The extended experiment is denoted by **S(L+IK)+P**. Fig 4.7 shows the quantitative results. The overall success rate of **S(L+IK)+P** is 44.7%/21.1% in the

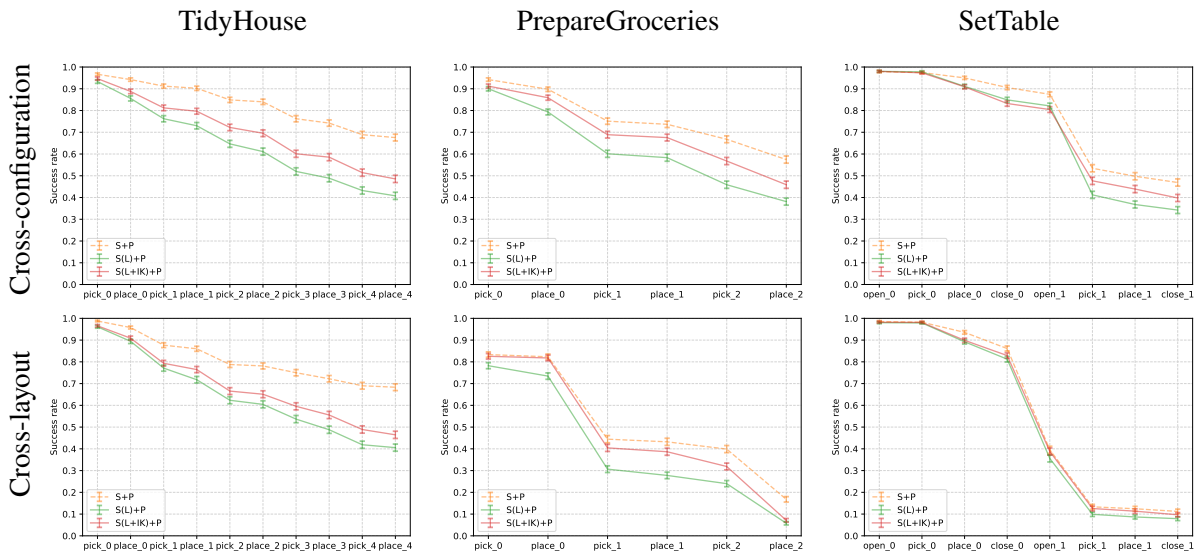


Figure 4.7. Progressive completion rates for HAB [109] tasks. The x-axis represents progressive subtasks. The y-axis represents the completion rate of each subtask. Results of ablation experiments are presented with solid lines. The mean and standard error for 100 episodes over 9 seeds are reported.

cross-configuration/cross-layout setting. It indicates that increasing the feasible initial states help stationary manipulation skills compared to **S(L)+P** (37.7%/18.1%), but still has a large performance drop compared to **S+P** (57.4%/31.1%). One possible reason is that although the target might be IK-reachable, it can be hard to achieve with stationary manipulation skills due to collision with other objects. However, mobile manipulation skills can first navigate to better locations with fewer obstacles in the front.

4.6 More Experiment Details

4.6.1 Dataset and Episodes

[109] keeps updating the ReplicaCAD dataset. The major fix is "minor furniture layout modifications in order to better accommodate robot access to the full set of receptacles" [6]. The agent radius is also decreased from 0.4m to 0.3m to generate navigation meshes with higher connectivity. Besides, [109] also improves the episode generator [7] to ensure stable initialization

[6] https://github.com/facebookresearch/habitat-sim/pull/1694
[7] https://github.com/facebookresearch/habitat-lab/pull/764

of objects. Those improvements eliminate most unachievable episodes in the initial version. The episodes used in our experiments are generated with the ReplicaCAD v1.4 and the latest habitat-lab [8].

Cross-configuration and cross-layout settings are the same except for scene layouts. In the cross-configuration setting, test scene layouts (micro variations) are different but similar to training ones. In the cross-layout setting, test scene layouts (macro variations) are significantly different from training ones. Each macro variation has a different, semantically plausible layout of large furniture (e.g., kitchen counter and fridge) while each micro variation is generated through perturbing small furniture (e.g., chairs and tables). Thus, the cross-layout setting demands stronger generalization on scene layouts.

For *TidyHouse*, each episode includes 20 clutter objects and 5 target objects along with their goal positions, located at 7 different receptacles (chair, 2 tables, tv stand, two kitchen counters, sofa). For *PrepareGroceries*, each episode includes 21 clutter objects located at 8 different receptacles (the 7 receptacles used in *TidyHouse* and the top shelf of the fridge) and 1 clutter object located at the middle shelf of the fridge. 2 target objects are located at the middle shelf, and each of their goal positions is located at one of two kitchen counters. The third target object is located at one of two kitchen counters, and its goal position is at the middle shelf. *SetTable* generates episodes similar to *PrepareGroceries*, except that two target objects, bowl and apple, are initialized at one of 3 drawers and at the middle fridge shelf respectively. Each of their goal positions is located at one of two tables.

4.6.2 Skill Learning

Each skill is trained to accomplish a subtask and reset its end-effector at the resting position. The robot arm is first initialized with predefined resting joint positions, such that the corresponding resting position of the end-effector is $(0.5, 1.0, 0.0)$ in the base frame [9]. The initial

[8] https://github.com/facebookresearch/habitat-lab/pull/837

[9] The positive x and y axes point forward and upward in Habitat.

end-effector position is then perturbed by a Gaussian noise $\mathcal{N}(0, 0.025)$ clipped at $0.05m$. The base position is perturbed by a Gaussian noise $\mathcal{N}(0, 0.1)$ truncated at $0.2m$. The base orientation is perturbed by a Gaussian noise $\mathcal{N}(0, 0.25)$ truncated at 0.5 radian. The maximum episode length is 200 steps for all the manipulation skills, and 500 steps for the navigation skill. The episode terminates on success or failure. We use the same reward function for both stationary and mobile manipulation skills, unless specified.

For all skills, d^o_{ee} is the distance between the end-effector and the object, d^r_{ee} is the distance between the end-effector and the resting position, d^h_{ee} is the distance between the end-effector and a predefined manipulation handle (a 3D position) of the articulated object, d^g_a is the distance between the joint position of the articulated object and the goal joint position. $\Delta^b_a = d^b_a(t-1) - d^b_a(t)$ stands for the (negative) change in distance between a and b. For example, Δ^o_{ee} is the change in distance between the end-effector and the object. $\mathbb{I}_{holding}$ indicates if the robot is holding an (correct) object or handle. $\mathbb{I}_{succ}$ indicates the task success. C_t refers to the current collision force, and $C_{1:t}$ stands for the accumulated collision force.

The 7-dim arm action stands for the delta joint positions added to the current target joint positions of the PD controller. The input arm action is assumed to be normalized to $[-1, 1]$, and will be scaled by 0.025 (radian). The 2-dim base action stands for linear and angular velocities. The base movement in the Habitat 2.0 is implemented by kinematically setting the robot's base transformation. The collision between the robot base and navigation meshes is taken into consideration. The input base action is assumed to be normalized to $[-1, 1]$, and will be scaled by 3 (navigation skill) or 1.5 (manipulation skills). For the navigation skill, we follow [109] to use a discrete action space and translate the discrete action into the continuous one. Concretely, the (normalized) linear velocity from -0.5 to 1 is discretized into 4 choices ($\{-0.5, 0, 0.5, 1\}$), and the (normalized) angular velocity from -1 to 1 is discretized into 5 choices (($\{-1, -0.5, 0, 0.5, 1\}$). The stop action corresponds to the discrete action representing zero velocities.

Pick(s_0)

- Objective: pick the object initialized at s_0
- Initial base position (noise is applied in addition):
  - Stationary: the closest navigable position to s_0
  - Mobile: a randomly selected navigable position within 2m of s_0
- Reward: $\mathbb{I}_{pick}$ indicates whether the correct object is picked and $\mathbb{I}_{wrong}$ indicates whether a wrong object is picked.

$$r_t = 4\Delta^o_{ee}\mathbb{I}_{!holding} + \mathbb{I}_{pick} + 4\Delta^r_{ee}\mathbb{I}_{holding} + 2.5\mathbb{I}_{succ}$$
$$- \max(0.001C_t, 0.2) - \mathbb{I}_{[C_{1:t}>5000]} - \mathbb{I}_{wrong} - \mathbb{I}_{[d^o_{ee}>0.09]}\mathbb{I}_{holding} - 0.002$$

- Success: The robot is holding the target object and the end-effector is within 5cm of the resting position. $\mathbb{I}_{succ} = \mathbb{I}_{holding} \wedge d^r_{ee} \leq 0.05$
- Failure:
  - $\mathbb{I}_{[C_{1:t}>5000]} = 1$: The accumulated collision force is larger than $5000N$.
  - $\mathbb{I}_{wrong} = 1$: A wrong object is picked.
  - $\mathbb{I}_{[d^o_{ee}>0.09]}\mathbb{I}_{holding} = 1$: The held object slides off the gripper.
- Observation space:
  - Depth images from head and arm cameras.
  - The current arm joint positions.
  - The current end-effector position in the base frame.
  - Whether the gripper is holding anything.
  - The starting position s_0 in both the base and end-effector frame.

- Action space: The gripper is disabled to release.

Place(s_*)

- Objective: place the held object at s_*
- Initial base position (noise is applied in addition):
  - Stationary: the closest navigable position to s_*
  - Mobile: a randomly selected navigable position within 2m of s_*
- Reward: $\mathbb{I}_{place}$ indicates whether the object is released within 15cm of the goal position, and $\mathbb{I}_{drop}$ indicates whether the object is released beyond 15cm.

$$r_t = 4\Delta_o^{s_*}\mathbb{I}_{holding} + \mathbb{I}_{place} + 4\Delta_{ee}^{r}\mathbb{I}_{!holding} + 2.5\mathbb{I}_{succ}$$
$$- \min(0.001C_t, 0.2) - \mathbb{I}_{[C_{1:t}>7500]} - \mathbb{I}_{drop} - \mathbb{I}_{[d_{ee}^{o}>0.09]}\mathbb{I}_{holding} - 0.002$$

- Success: The object is within 15cm of the goal position and the end-effector is within 5cm of the resting position. $\mathbb{I}_{succ} = d_o^{s_*} \leq 0.15 \wedge \mathbb{I}_{!holding} \wedge d_{ee}^{r} \leq 0.05$
- Failure:
  - $\mathbb{I}_{[C_{1:t}>7500]} = 1$: The accumulated collision force is larger than $7500N$.
  - $\mathbb{I}_{drop} = 1$: The object is released beyond 15cm of the goal position.
  - $\mathbb{I}_{[d_{ee}^{o}>0.09]}\mathbb{I}_{holding} = 1$: The held object slides off the gripper.
- Observation space:
  - Depth images from head and arm cameras.
  - The current arm joint positions.
  - The current end-effector position in the base frame.
  - Whether the gripper is holding anything.

  - The goal position s_* in both the base and end-effector frame.

- Action space: The gripper is disabled to grasp after releasing the object.

Open drawer(s)

- Objective: open the drawer containing the object initialized at s. The goal joint position of the drawer is $g = 0.45m$.
- Initial base position (noise is applied in addition):
  - Stationary: a navigable position randomly selected within a $[0.80, -0.35] \times [0.95, 0.35]$ region in front of the drawer.
  - Mobile: a navigable position randomly selected within a $[0.3, -0.6] \times [1.5, 0.6]$ region in front of the drawer.
- Reward: $\mathbb{I}_{open} = d_a^g \leq 0.05$ indicates whether the drawer is open. $\mathbb{I}_{release}$ indicates whether the handle is released when the drawer is open. $\mathbb{I}_{grasp}$ indicates whether the correct handle is grasped. a_{base} is the (2-dim) base action.

$$r_t = 2\Delta_{ee}^h \mathbb{I}_{!open} + \mathbb{I}_{grasp} + 2\Delta_a^g \mathbb{I}_{holding} + \mathbb{I}_{release} + 2\Delta_{ee}^r \mathbb{I}_{open} + 2.5\mathbb{I}_{succ} \\ -\mathbb{I}_{wrong} - \mathbb{I}_{[d_{ee}^h > 0.2]} \mathbb{I}_{holding} - \mathbb{I}_{out} - 0.004\|a_{base}\|_1$$

- Success: The drawer is open, and the end-effector is within 15cm of the resting position. $\mathbb{I}_{succ} = \mathbb{I}_{open} \wedge \mathbb{I}_{!holding} \wedge d_{ee}^r \leq 0.15$
- Failure:
  - $\mathbb{I}_{wrong} = 1$: The wrong object or handle is picked.
  - $\mathbb{I}_{[d_{ee}^h > 0.2]} \mathbb{I}_{holding} = 1$: The grasped handle slides off the gripper.
  - $\mathbb{I}_{out} = 1$: The robot moves out of a predefined region (a $2m \times 3m$ region in front of the drawer).

- $\mathbb{I}_{\mathbb{I}_{[open(t-1)\wedge !open(t)]}} = 1$: The drawer is not open after being opened.
- The gripper releases the handle when the drawer is not open ($\mathbb{I}_{!open} = 1$).
- $\Delta_a^g >= 0.1$: The drawer is opened too fast.

- Observation space:
  - Depth images from head and arm cameras.
  - The current arm joint positions.
  - The current end-effector position in the base frame.
  - Whether the gripper is holding anything.
  - The starting position s in both the base and end-effector frame.

Close drawer(s)

- Objective: close the drawer containing the object initialized at s. The goal joint position is $g = 0m$.
- Initial joint position: $q_a \in [0.4, 0.5]$, where q_a is the joint position of the target drawer. A random subset of other drawers are slightly open ($q'_a \leq 0.1$).
- Initial base position (noise is applied in addition):
  - Stationary: a navigable position randomly selected within a $[0.3, -0.35] \times [0.45, 0.35]$ region in front of the drawer.
  - Mobile: a navigable position randomly selected within a $[0.3, -0.6] \times [1.0, 0.6]$ region in front of the drawer.
- Reward: It is almost the same as *Open drawer* by replacing *open* with *close*. $\mathbb{I}_{close} = d_a^g \leq 0.1$.
- Success: The drawer is closed, and the end-effector is within 15cm of the resting position.
- Failure: It is almost the same as *Open drawer* by replacing *open* with *close*, except that the last constraint $\Delta_a^g >= 0.1$ is not included.

Open fridge(s)

- Objective: open the fridge containing the object initialized at s. The goal joint position is $g = \frac{\pi}{2}$.
- Initial base position (noise is applied in addition): a navigable position randomly selected within a $[0.933, -1.5] \times [1.833, 1.5]$ region in front of the fridge.
- Reward: $\mathbb{I}_{open} = g - q_a > 0.15$, where q_a is the joint position (radian) of the fridge. To avoid the robot from penetrating the fridge due to simulation defects, we add a collision penalty but excludes collision between the end-effector and the fridge.

$$r_t = 2\Delta^h_{ee}\mathbb{I}_{!open} + \mathbb{I}_{grasp} + +2\Delta^g_a\mathbb{I}_{holding} + \mathbb{I}_{release} + \Delta^r_{ee}\mathbb{I}_{open} + 2.5\mathbb{I}_{succ}$$
$$-\mathbb{I}_{C_{1:t}>5000} - \mathbb{I}_{wrong} - \mathbb{I}_{[d^h_{ee}>0.2]}\mathbb{I}_{holding} - \mathbb{I}_{out} - 0.004\|a_{base}\|_1$$

- Success: The fridge is open, and the end-effector is within 15cm of the resting position. $\mathbb{I}_{succ} = \mathbb{I}_{open} \wedge \mathbb{I}_{!holding} \wedge d^r_{ee} \leq 0.15$
- Failure:
  - $\mathbb{I}_{wrong} = 1$: The wrong object or handle is picked.
  - $\mathbb{I}_{[d^h_{ee}>0.2]}\mathbb{I}_{holding} = 1$: The grasped handle slides off the gripper.
  - $\mathbb{I}_{out} = 1$: The robot moves out of a predefined region (a $2m \times 3.2m$ region in front of the fridge).
  - $\mathbb{I}_{\mathbb{I}_{[open(t-1)\wedge !open(t)]}} = 1$: The fridge is not open after being opened.
  - The gripper releases the handle when the fridge is not open ($\mathbb{I}_{!open} = 1$).
- Observation space:
  - Depth images from head and arm cameras.
  - The current arm joint positions.

  - The current end-effector position in the base frame.
  - Whether the gripper is holding anything.
  - The starting position s in both the base and end-effector frame.

Close fridge(s)

- Objective: close the fridge containing the object initialized at s. The goal joint position is $g = 0$.
- Initial joint position: $q_a \in [\frac{\pi}{2} - 0.15, 2.356]$, where q_a is the joint position of the target fridge.
- Initial base position (noise is applied in addition): a navigable position randomly selected within a $[0.933, -1.5] \times [1.833, 1.5]$ region in front of the fridge.
- Reward: It is almost the same as *Close fridge* by replacing *open* with *close*. $\mathbb{I}_{close} = d_a^g \leq 0.15$.
- Success: The fridge is close, and the end-effector is within 15cm of the resting position.

Navigate(s) (point-goal)

- Objective: navigate to the start of other skills specified by s
- Reward: refer to Eq 4.1. $r_{slack} = 0.002, \tilde{D} = 0.9, \lambda_{ang} = 0.25, \lambda_{succ} = 2.5$
- Success: The robot is within 0.3 meter of the goal, 0.5 radian of the target orientation, and has called the stop action at the current time step.
- Observation space:
  - Depth images from the head camera.
  - The goal position s_* in the base frame.

Navigate(s) (region-goal)

- Objective: navigate to the start of other skills specified by s
- Reward: refer to Eq 4.2. $r_{slack} = 0.002, r_{col} = \min(0.001C_t, 0.2), \lambda_{succ} = 2.5$
- Success: The robot is within 0.1 meter of any goal in the region, 0.25 radian of the target orientation at the current position, and has called the stop action at the current time step.
- Observation space:
  - Depth images from the head camera.
  - The goal position s_* in the base frame.

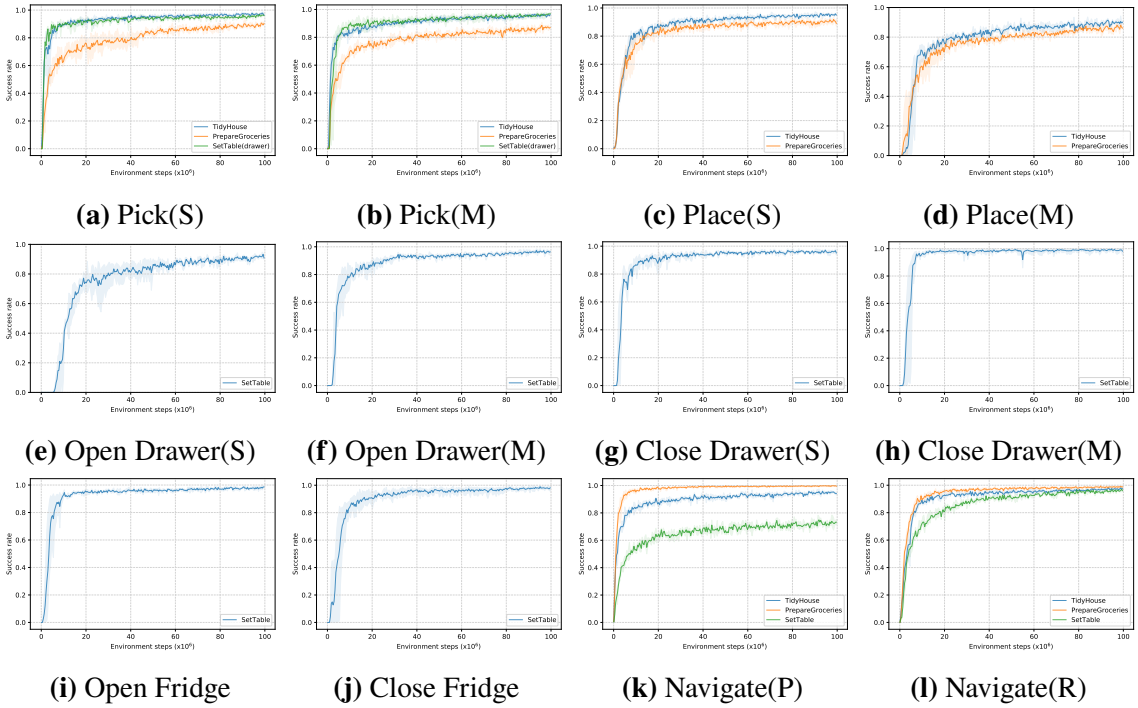
4.6.3 PPO Hyper-parameters

Our PPO implementation is based on the habitat-lab. The visual encoder is a simple CNN [10]. The coefficients of value and entropy losses are 0.5 and 0 respectively. We use 64 parallel environments and collect 128 transitions per environment to update the networks. We use 2 mini-batches, 2 epochs per update, and a clipping parameter of 0.2 for both policy and value. The gradient norm is clipped at 0.5. We use the Adam optimizer with a learning rate of 0.0003. The linear learning rate decay is enabled. The mean of the Gaussian action predicted by the policy network is activated by tanh. The (log) standard deviation of the Gaussian action, which is an input-independent parameter, is initialized as -1.0. Fig 4.8 shows training curves of skills.

4.6.4 Other Implementation Details

The PPO algorithm implemented by the habitat-lab does not distinguish the termination of the environment (MDP) and the truncation due to time limit. We fix this issue in our implementation. Furthermore, we separately train all the skills for each HAB task to avoid potential ambiguity. For example, the starting position of an object in the drawer is computed when the drawer is closed at the beginning of an episode. However, the skill *Pick* needs to pick

[10] https://github.com/facebookresearch/habitat-lab/blob/main/habitat_baselines/rl/models/simple_cnn.py



(a) Pick(S) **(b)** Pick(M) **(c)** Place(S) **(d)** Place(M)

(e) Open Drawer(S) **(f)** Open Drawer(M) **(g)** Close Drawer(S) **(h)** Close Drawer(M)

(i) Open Fridge **(j)** Close Fridge **(k)** Navigate(P) **(l)** Navigate(R)

Figure 4.8. Training curves for skills. The y-axis represents the success rate of the subtask (including resetting the end-effector at its resting position). "S" and "M" stand for stationary and mobile manipulation skills. "P" and "R" stand for point- and region-goal navigation skills. Best viewed zoomed.

this object up when the drawer is open and the actual position of the object is different from the starting position. It is inconsistent with other cases when the object is in an open receptacle or the fridge. We observe such ambiguity can hurt performance. See Fig 4.8 for all task-specific variants of skills.

4.6.5 Monolithic Baseline

For the monolithic baseline, a monolithic RL policy is trained for each HAB task. During training, the policy only handles one randomly selected target object, *e.g.*, picking and placing one object in *TidyHouse*. During inference, the policy is applied to each target object. We use the same observation space, action space and training scheme as those for our mobile manipulation skills. The main challenge is how to formulate a reward function for those complicated long-horizon HAB tasks that usually require multiple stages. We follow [109] to composite reward functions

for individual skills, given the sequence of subtasks. Concretely, at each time step during training, we infer the current subtask given perfect knowledge of the environment, and use the reward function of the corresponding skill. To ease training, we remove the collision penalty and do not terminate the episode due to collision. Besides, we use the region-goal navigation reward for the navigation subtask. Thanks to our improved reward functions and better training scheme, our monolithic RL baseline is much better than the original implementation in [109]. However, although able to move the object to its goal position, the policy never learns to release the object to complete the subtask *Place* during training. It might be due to exploration difficulty since *Place* is the last subtask in a long sequence and previous subtasks all require the robot not to release. To boost its performance, we force the gripper to release anything held at the end of execution during evaluation.

4.7 More Evaluation Details

4.7.1 Sequential Skill Chaining

For evaluation, skills are sequentially executed in the order of their corresponding subtasks, as described in Sec 4.3.3. The main challenge is how to terminate a skill without privileged information. Basically, each skill will be terminated if its execution time exceeds its max episode length (200 steps for manipulation skills and 500 steps for the navigation skill). The termination condition of *Pick* is that an object is held and the end-effector is within 15cm of the resting position, which can be computed based on proprioception only. The gripper is disabled to release for *Pick*. The termination condition of *Place* is that the gripper holds nothing and the end-effector is within 15cm of the resting position. The gripper is disabled to grasp for *Place*. Besides, anything held will be released when *Place* terminates. For *Open* and *Close*, we use a heuristic from [109]: the skill will terminate if the end-effector is within 15cm of the resting position and it has moved at least 30cm away from the resting position during execution. *Navigate* terminates when it calls the stop action. Furthermore, since the manipulation skills only learn to reset

its end-effector, we apply an additional operation to reset the whole arm after each skill. This reset operation is achieved by setting predefined joint positions as the target of the robot's PD controller.

4.7.2 Progressive Completion Rate

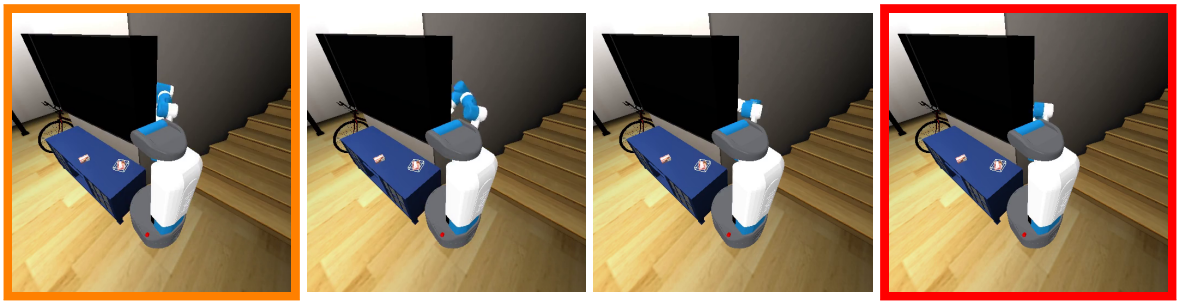
In this section, we describe how progressive completion rates are computed. The evaluation protocol is the same as [109] (see its Appendix F), and here we phrase it in a way more friendly to readers with little knowledge of task planning and Planning Domain Definition Language (PDDL). To partially evaluate a HAB task, we divide a full task into a sequence of stages (subgoals). For example, *TidyHouse* can be considered to consist of *pick_0*, *place_0*, *pick_1*, *etc*.Each stage can correspond to multiple subtasks. For example, the stage *pick_i* includes *Navigate*(s_0^i) and *Pick*(s_0^i). Thus, to be precise, the completion rate is computed based on stages instead of subtasks. We define a set of predicates to measure whether the goal of a stage is completed. A stage goal is completed if all the predicates associated with it are satisfied. The predicates are listed as follows:

- `holding(target_obj|i)`: The robot is holding the i-th object.
- `at(target_obj_pos|i,target_goal_pos|i)`: The i-th object is within 15cm of its goal position.
- `opened_drawer(target_marker|i)`: The target drawer is open (the joint position is larger than 0.4m).
- `closed_drawer(target_marker|i)`: The target drawer is close (the joint position is smaller than 0.1m).
- `opened_fridge(target_marker|i)`: The target fridge is open (the joint position is larger than $\frac{\pi}{2}$ radian).
- `closed_fridge(target_marker|i)`: The target fridge is close (the joint position is smaller than 0.15 radian).

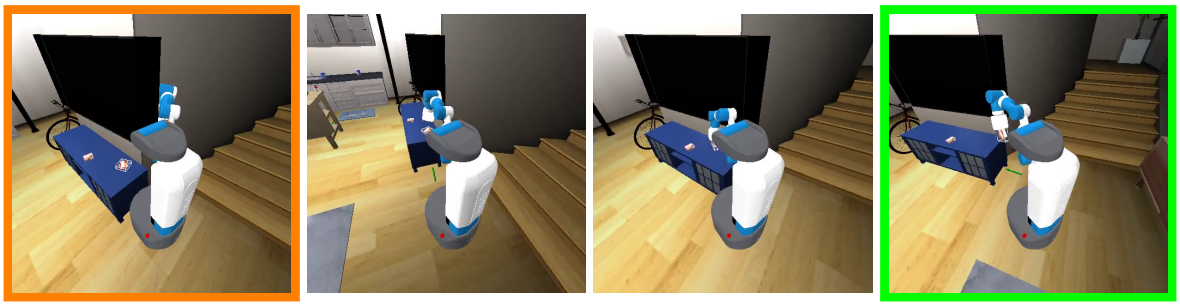
During evaluation, we evaluate whether the current stage goal is completed at each time step. If the current stage goal is completed, we progress to the next stage. Hence, the completion rate monotonically decreases. Listings 4.1, 4.2, 4.3 present the stages defined for each HAB task and the predicates associated with each stage. Note that the stage goal *place_i* only indicates that the object has been released at its goal position, but the placement can be unstable (*e.g.*, the object falls down the table), which can lead to the failure of the next stage. Besides, due to abstract grasp, it is difficult to place the object stably since the pose of the grasped object can not be fully controlled. Therefore, we modify the objective of *SetTable* to make the task achievable given abstract grasp. Concretely, instead of placing the fruit in the bowl, the robot only needs to place the fruit picked from the fridge at a goal position on the table.

4.8 More Qualitative Results

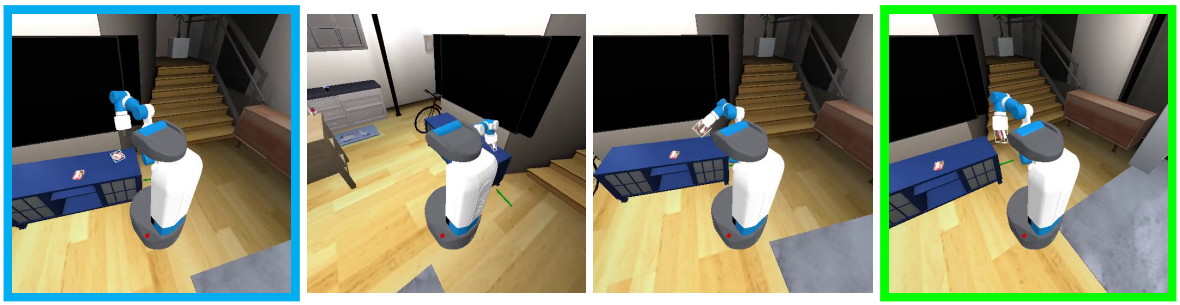
Fig 4.9, 4.10, 4.11 show more qualitative comparison of different methods. Their animated versions can be found on our project website.



(a) Stationary manipulation and point-goal navigation (S+P)

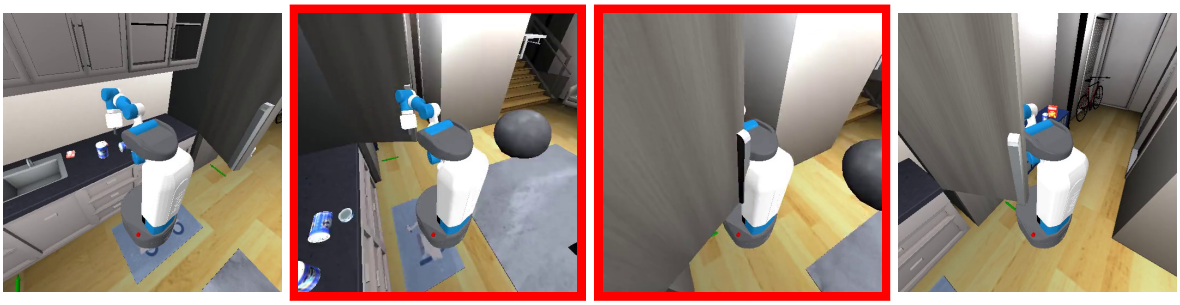


(b) Mobile manipulation and point-goal navigation (M+P)

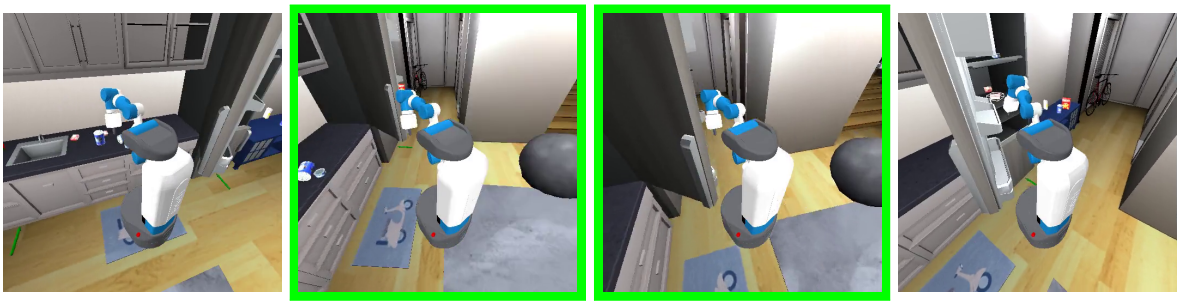


(c) Mobile manipulation and region-goal navigation (M3)

Figure 4.9. Qualitative comparison in *TidyHouse*. In this example, the point-goal navigation skill terminates behind the TV (1st image). The arm is blocked by the TV in stationary manipulation (last image in the top row). The robot manages to move backward and avoid being blocked in mobile manipulation (last image in the middle row). The region-goal navigation skill instead terminates in front of the TV (1st image in the bottom row).

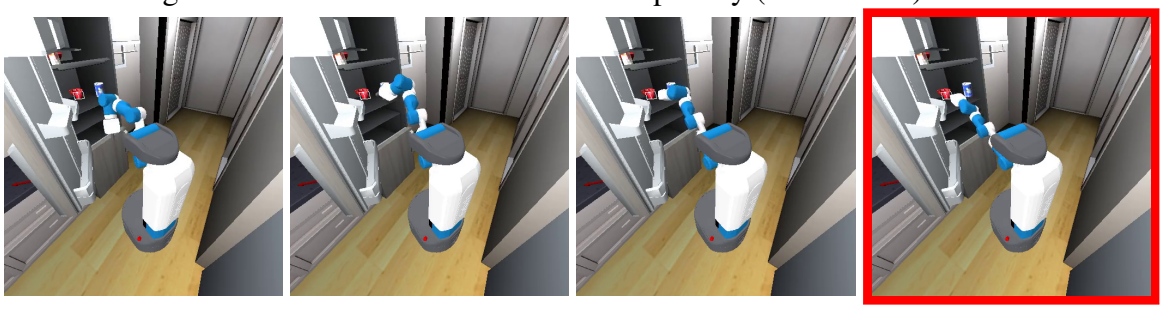


(a) Point-goal navigation



(b) Region-goal navigation

Figure 4.10. Qualitative comparison in *PrepareGroceries*. In this example, the point-goal navigation skill accidentally close the fridge (top row). The region-goal navigation skill is able to avoid disturbing the environment due to the collision penalty (bottom row).



(a) Stationary manipulation

(b) Mobile manipulation

Figure 4.11. Qualitative comparison in *SetTable*. In this example, the navigation skill terminates at the position where the robot can not reach the target object in the fridge in stationary manipulation. (top row). The robot can move closer to the object and then pick it, to compensate for the navigation skill in mobile manipulation (bottom row).

Listing 4.1. Stage goals and their associated predicates defined for *TidyHouse*. The stages are listed in the order for progressive evaluation.

```
pick_0:
- "holding(target_obj|0)"
place_0:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
pick_1:
- "holding(target_obj|1)"
- "at(target_obj_pos|0,target_goal_pos|0)"
place_1:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
pick_2:
- "holding(target_obj|2)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
place_2:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
- "at(target_obj_pos|2,target_goal_pos|2)"
pick_3:
- "holding(target_obj|3)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
- "at(target_obj_pos|2,target_goal_pos|2)"
place_3:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
- "at(target_obj_pos|2,target_goal_pos|2)"
- "at(target_obj_pos|3,target_goal_pos|3)"
pick_4:
- "holding(target_obj|4)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
- "at(target_obj_pos|2,target_goal_pos|2)"
- "at(target_obj_pos|3,target_goal_pos|3)"
place_4:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
- "at(target_obj_pos|2,target_goal_pos|2)"
- "at(target_obj_pos|3,target_goal_pos|3)"
- "at(target_obj_pos|4,target_goal_pos|4)"
```

Listing 4.2. Stage goals and their associated predicates defined for *PrepareGroceries*. The stages are listed in the order for progressive evaluation.

```
pick_0:
- "holding(target_obj|0)"
place_0:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
pick_1:
- "holding(target_obj|1)"
- "at(target_obj_pos|0,target_goal_pos|0)"
place_1:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
pick_2:
- "holding(target_obj|2)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
place_2:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "at(target_obj_pos|1,target_goal_pos|1)"
- "at(target_obj_pos|2,target_goal_pos|2)"
```

Listing 4.3. Stage goals and their associated predicates defined for *SetTable*. The stages are listed in the order for progressive evaluation.

```
open_0:
- "opened_drawer(target_marker|0)"
pick_0:
- "holding(target_obj|0)"
place_0:
- "not_holding()"
- "at(target_obj_pos|0,target_goal_pos|0)"
close_0:
- "closed_drawer(target_marker|0)"
- "at(target_obj_pos|0,target_goal_pos|0)"
open_1:
- "closed_drawer(target_marker|0)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "opened_fridge(target_marker|1)"
pick_1:
- "closed_drawer(target_marker|0)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "opened_fridge(target_marker|1)"
- "holding(target_obj|1)"
place_1:
- "closed_drawer(target_marker|0)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "not_holding()"
- "at(target_obj_pos|1,target_goal_pos|1)"
close_1:
- "closed_drawer(target_marker|0)"
- "at(target_obj_pos|0,target_goal_pos|0)"
- "closed_fridge(target_marker|1)"
- "at(target_obj_pos|1,target_goal_pos|1)"
```

4.9 Conclusion and Limitations

In this work, we present a modular approach to tackle long-horizon mobile manipulation tasks in the Home Assistant Benchmark (HAB), featuring mobile manipulation skills and the region-goal navigation reward. Given the superior performance, our approach can serve as a strong baseline for future study. Besides, the proposed principles (*achievability, composability, reusability*) can serve as a guideline about how to formulate meaningful and reusable subtasks. However, our work is still limited to abstract grasp and other potential simulation defects. We leave fully dynamic simulation and real-world deployment to future work.

Acknowledgement

Chapter 4, in full, is a reprint of the material published in the 2023 International Conference on Learning Representations (ICLR): "Multi-skill Mobile Manipulation for Object Rearrangement" (Jiayuan Gu, Devendra Singh Chaplot, Hao Su, Jitendra Malik). The dissertation author was the primary investigator and author of this paper.

Chapter 5

Finale

In this dissertation, we introduce our work on learning generalist robot manipulation policies, an endeavor that unfolds across three key areas: data, model, and task. To collect diverse demonstration data and evaluate policies, we developed a simulated benchmark named *ManiSkill2*, designed to foster generalizable manipulation skills. To enhance task-level generalization, we propose the use of trajectory sketches within *RT-Trajectory*, which enables visual prompting for robot policies. In our study on *Multi-skill Mobile Manipulation (M3)*, we investigate which tasks are more conducive to mobile manipulation, in the context of skill chaining.

Despite the remarkable progress in the field, many important questions remain unanswered. Here, I list several key issues that I am eager to explore in my future work:

1. Sim-to-Real for Generic Manipulation Policies: While sim-to-real techniques have seen success in locomotion tasks for quadrupeds and humanoids, their application in manipulation tasks involving rich contacts and complex dynamics, or even generic pick-and-place tasks, remains challenging. Such manipulation tasks, despite appearing simple sometimes, depend on interactions between robots and a wide variety of objects, unlike locomotion tasks that mainly focus on the robot itself. The primary hurdle lies in the vast diversity of objects that can be manipulated. The diversity of objects results in diverse visual appearances and various ways to manipulate. It necessitates scalable methods for generating a wide array of assets including objects, scenes, and their layouts in simulated environments.

This diversity is key to bridging the sim-to-real gap, suggesting that improvements in diversity and scale could naturally narrow this gap. Furthermore, manipulation hardware often features lower degrees of freedom (e.g., 7 DoF for a single arm and 2 DoF for a parallel gripper) compared to locomotion hardware (typically 20+ DoF). Similar to neural networks, over-parameterization can facilitate finding suboptimal yet satisfactory solutions, whereas under-parameterization presents challenging scenarios that might not always be necessary to address. It remains uncertain whether the techniques we develop for current hardware will be necessary and useful for future hardware advancements.

2. Real-to-Sim for Evaluation: Evaluating robot policies in the real world is notoriously difficult, time-consuming, costly, and hard to replicate. Simulations offer a valuable alternative, but the creation of realistic simulated environments that accurately mirror real-world scenarios is a significant challenge. This task calls upon advanced techniques from the graphics community, such as inverse rendering and physical simulation, to achieve high fidelity in simulated environments.

3. Data Curation for Generalist Policies: A critical question in developing generalist robot manipulation policies is deciding which skills to focus on and how to organize heterogeneous demonstrations that span different observation and action spaces. Like curating a dataset akin to ImageNet for the robotics domain, the management and organization of these datasets are paramount. This includes considering the types of skills valuable for a generalist policy, and how to structure and annotate the data to support effective learning.

4. Cross-Embodiment Robot Policies: The issue of embodiment in robot manipulation policies is crucial, as it significantly influences how a robot interacts with its environment. This necessitates the development of policies that are not only effective but also flexible enough to adapt to various robotic platforms. However, the necessity of creating policies that span across different embodiments is debatable. For instance, humans do not share manipulation policies with animals such as dogs or cats. This might suggest that a universally applicable

cross-embodiment policy is not essential. Nevertheless, it's possible to argue that humans possess a kind of foundational model, perhaps coded in their genes, which allows for online adaptation to their unique physical forms. An analogy can be drawn to a person maturing: as they grow, they must "recalibrate" their understanding and control of their body to perform tasks they were previously familiar with. This concept underscores the potential for designing adaptable and learning-centric robot policies that can adjust to the specificities of their embodiment, much like humans adapt to theirs over time.

5. Combining High-Level Planning with Low-Level Skills: Robots, akin to the dual-process theory of human cognition "System 1 and System 2 thinking" [55], may embody similar modes of operation. System 1—representative of low-level skills—is fast and instinctive, while System 2—aligned with high-level planning—is slower, more deliberative, and logical. It introduces a unique set of challenges and questions. For instance, high-level planning has seen advancements through the use of Large Language Models (LLMs), which are capable of generating sophisticated reasoning steps. However, there remains a gap as our low-level policies often struggle to fully execute these high-level instructions.

   Furthermore, a common limitation of current low-level policies is their inability to provide feedback to the high-level planner, particularly regarding their capacity to address given tasks. This issue was approached by the introduction of affordance prediction in the Say-Can [1] framework, which aims to bridge this communication gap. This area presents ample opportunity for exploration, suggesting that further integration of high-level reasoning and low-level execution could enhance adaptability and efficiency of robotic systems.

Bibliography

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

[2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

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

[4] Dhruv Batra, Angel X. Chang, S. Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020.

[5] Kent L. Beck, Michael A. Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andy Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, and Dave A. Thomas. Manifesto for agile software development, 2001.

[6] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. Data quality in imitation learning. *arXiv preprint arXiv:2306.02437*, 2023.

[7] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel J. Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[8] Gunilla Borgefors. Distance transformations in arbitrary dimensions. *Computer vision, graphics, and image processing*, 27(3):321–345, 1984.

[9] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X. Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, Antoine Laurens, Claudio Fantacci, Valentin Dalibard, Martina Zambelli, Murilo Martins, Rugile Pevceviciute, Michiel Blokzijl, Misha Denil, Nathan Batchelor, Thomas Lampe, Emilio Parisotto, Konrad Żo lna, Scott Reed, Sergio Gómez Colmenarejo, Jon Scholz, Abbas Abdolmaleki, Oliver Groth, Jean-Baptiste Regli, Oleg Sushkov, Tom Rothörl, José Enrique Chen, Yusuf Aytar, Dave Barker, Joy Ortiz, Martin Riedmiller, Jost Tobias Springenberg, Raia Hadsell, Francesco Nori, and Nicolas Heess. Robocat: A self-improving foundation agent for robotic manipulation, 2023.

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

[11] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence,

Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.

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

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[14] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.

[15] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Learning video-conditioned policies for unseen manipulation tasks, 2023.

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

[17] Zoey Chen, Sho Kiami, Abhishek Gupta, and Vikash Kumar. Genaug: Retargeting

behaviors to unseen situations via generative augmentation. *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[18] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[19] Alexander Clegg, Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Learning to dress: Synthesizing human dressing motion via deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(6):1–10, 2018.

[20] Hilko Cords and Oliver G. Staadt. Instant liquids: Fast screen-space surface generation for particle-based liquids. In *Poster proceedings of ACM Siggraph/Eurographics Symposium on Computer Animation*, 2008.

[21] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[22] Murtaza Dalal, Ajay Mandlekar, Caelan Reed Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers. In *Conference on Robot Learning*, 2023.

[23] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023.

[24] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4495–4504, 2021.

[25] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. 1994.

[26] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. Acronym: A large-scale grasp dataset based on simulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6222–6227. IEEE, 2021.

[27] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[28] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[29] M Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1):1–72, 1906.

[30] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Damian Mrowca, Michael Lingelbach, Aidan Curtis, Kevin T. Feigelis, Daniel Bear, Dan Gutfreund, David Cox, James J. DiCarlo, Josh H. McDermott, Joshua B. Tenenbaum, and Daniel L. K. Yamins. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

[31] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel L. K. Yamins, James J. DiCarlo, Josh H. McDermott, Antonio Torralba, and Joshua B. Tenenbaum. The threedworld transport challenge: A visually guided task-and-motion planning benchmark towards physically realistic embodied ai. *2022 International Conference on Robotics and Automation (ICRA)*, pages 8847–8854, 2021.

[32] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[33] 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.

[34] Ran Gong, Jiangyong Huang, Yizhou Zhao, Haoran Geng, Xiaofeng Gao, Qingyang Wu, Wensi Ai, Ziheng Zhou, Demetri Terzopoulos, Song-Chun Zhu, Baoxiong Jia, and Siyuan Huang. Arnold: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 20426–20438, 2023.

[35] Montserrat Gonzalez Arenas, Ted Xiao, Sumeet Singh, Vidhi Jain, Allen Ren, Quan Vuong, Jake Varley, Alexander Herzog, Isabel Leal, Sean Kirmani, Mario Prats, Dorsa Sadigh, Vikas Sindhwani, Kanishka Rao, Jacky Liang, and Andy Zeng. How to prompt your robot: A promptbook for manipulation skills with code as policies. In *IEEE international conference on robotics and automation (ICRA)*, 2024.

[36] Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill mobile manipulation for object rearrangement. In *The Eleventh International Conference on Learning Representations*, 2023.

[37] Jiayuan Gu, Han Hu, Liwei Wang, Yichen Wei, and Jifeng Dai. Learning region features for object detection. In *Proceedings of the european conference on computer vision (ECCV)*, pages 381–395, 2018.

[38] Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. In *The Eleventh International Conference on Learning Representations*, 2024.

[39] Jiayuan Gu, Wei-Chiu Ma, Sivabalan Manivasagam, Wenyuan Zeng, Zihao Wang, Yuwen Xiong, Hao Su, and Raquel Urtasun. Weakly-supervised 3d shape completion in the wild. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 283–299. Springer, 2020.

[40] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang 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 *The Eleventh International Conference on Learning Representations*, 2023.

[41] 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.

[42] Felix Hill, Sona Mokrá, Nathaniel Wong, and Tim Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *CoRR*, abs/2005.09382, 2020.

[43] Rachel M Holladay and Siddhartha S Srinivasa. Distance metrics and algorithms for task space path optimization. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5533–5540. IEEE, 2016.

[44] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3588–3597. IEEE, 2018.

[45] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):150, 2018.

[46] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[47] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models, 2023.

[48] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. *arXiv preprint arXiv:2104.03311*, 2021.

[49] 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.

[50] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning*, 2021.

[51] Maximilian Jaritz, Jiayuan Gu, and Hao Su. Multi-view pointnet for 3d scene understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

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

[53] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. In *Fortieth International Conference on Machine Learning*, 2023.

[54] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677, 2020.

[55] Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011.

[56] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[57] Dmitry Kalashnikov, Jake Varley, Yevgen Chebotar, Ben Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *Conference on Robot Learning (CoRL)*, 2021.

[58] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[59] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3992–4003, 2023.

[60] Youngwoon Lee, Joseph J. Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. In *Conference on Robot Learning*, 2021.

[61] 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*, 2018.

[62] Youngwoon Lee, Jingyun Yang, and Joseph J Lim. Learning to coordinate manipulation skills via skill behavior diversification. In *International Conference on Learning Representations*, 2019.

[63] Chengshu Li, Ruohan Zhang, J. Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R. Matthews, Ivan Villa-Renteria, Jerry Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. 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.

[64] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *arXiv preprint arXiv:2209.07753*, 2022.

[65] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 2020.

[66] Minghua Liu, Xuanlin Li, Zhan Ling, Yangyan Li, and Hao Su. Frame mining: a free lunch for learning robotic manipulation from 3d point clouds. In *6th Annual Conference on Robot Learning*, 2022.

[67] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua,

Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for perceiving and processing reality. In *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR)*, 2019.

[68] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. *arXiv preprint arXiv:2401.16013*, 2024.

[69] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *Conference on Robot Learning (CoRL)*, 2019.

[70] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*, 2021.

[71] Wei-Chiu Ma, Shenlong Wang, Jiayuan Gu, Sivabalan Manivasagam, Antonio Torralba, and Raquel Urtasun. Deep feedback inverse problem solver. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 229–246. Springer, 2020.

[72] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[73] 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. *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[74] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964. IEEE, 2016.

[75] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, N. Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[76] 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. *Conference on Robot Learning (CoRL)*, 2021.

[77] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018.

[78] Roberto Martín-Martín, Michelle A Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1010–1017. IEEE, 2019.

[79] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[80] Douglas Morrison, Peter Corke, and Jürgen Leitner. Egad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters*, 5(3):4368–4375, 2020.

[81] Tongzhou Mu, Jiayuan Gu, Zhiwei Jia, Hao Tang, and Hao Su. Refactoring policy for compositional generalizability using self-supervised object proposals. *Advances in Neural Information Processing Systems*, 33:8883–8894, 2020.

[82] 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.

[83] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation, 2021.

[84] Tianwei Ni, Kiana Ehsani, Luca Weihs, and Jordi Salvador. Towards disturbance-free visual mobile manipulation. *arXiv preprint arXiv:2112.12612*, 2021.

[85] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[86] Maxime Oquab, Timoth'ee Darcet, Théo Moutakanni, Huy Q. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russ Howes, Po-Yao (Bernie) Huang, Shang-Wen Li, Ishan Misra, Michael G. Rabbat, Vasu Sharma, Gabriel Synnaeve, Huijiao Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *arXiv preprint*

arXiv:2304.07193, 2023.

[87] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[88] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *IEEE international conference on robotics and automation (ICRA)*, 2016.

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

[90] Xavi Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondru*vs*., Théophile Gervet, Vincent-Pierre Berges, John Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Devendra Jitendra Malik, Singh Chaplot, Unnat Jain, Dhruv Batra, AksharaRai, and RoozbehMottaghi. Habitat 3.0: A co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.

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

[92] Haozhi Qi, Ashish Kumar, Roberto Calandra, Yinsong Ma, and Jitendra Malik. In-hand object rotation via rapid motor adaptation. In *Conference on Robot Learning*, 2022.

[93] 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.

[94] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[95] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[96] 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.

[97] IEEE RAS. Mobile manipulation. https://www.ieee-ras.org/mobile-manipulation, 2022. Accessed: 2022-05-18.

[98] Fetch Robotics. Autonomous mobile robots that improve productivity. http://fetchrobotics.com/, 2022. Accessed: 2022-05-18.

[99] Thushara Sandakalum and Marcelo H Ang Jr. Motion planning for mobile manipulators—a systematic review. *Machines*, 10(2):97, 2022.

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

[101] Martin Sereinig, Wolfgang Werth, and Lisa-Marie Faller. A review of the challenges in mobile manipulation: systems design and robocup challenges. *e & i Elektrotechnik und Informationstechnik*, 137(6):297–308, 2020.

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

[103] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[104] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Mart'in-Mart'in, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, S. 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. In *Conference on Robot Learning*, pages 477–490. PMLR, 2022.

[105] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[106] Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, and Karol Hausman. Open-world object manipulation using pre-trained vision-language models, 2023.

[107] Charles Sun, Jedrzej Orbik, Coline Manon Devin, Brian H Yang, Abhishek Gupta, Glen Berseth, and Sergey Levine. Fully autonomous real-world reinforcement learning with applications to mobile manipulation. In *Conference on Robot Learning*, pages 308–319. PMLR, 2022.

[108] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-

graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE, 2021.

[109] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[110] Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020.

[111] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, pages 23–30. IEEE, 2017.

[112] 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.

[113] Sam Toyer, Rohin Shah, Andrew Critch, and Stuart Russell. The magical benchmark for robust imitation. *Advances in Neural Information Processing Systems*, 33:18284–18295, 2020.

[114] 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.

[115] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[116] Cong Wang, Qifeng Zhang, Qiyan Tian, Shuo Li, Xiaohui Wang, David Lane, Yvan Petillot, and Sen Wang. Learning mobile manipulation through deep reinforcement learning. *Sensors*, 20(3):939, 2020.

[117] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. *Advances in neural information processing systems*, 31, 2018.

[118] Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition

for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022.

[119] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5922–5931, 2021.

[120] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *arXiv preprint arXiv:2107.14171*, 2021.

[121] Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zi-Yan Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. Envpool: A highly parallel reinforcement learning environment execution engine. *arXiv preprint arXiv:2206.10558*, 2022.

[122] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *International Conference on Learning Representations*, 2019.

[123] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[124] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[125] Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. *arXiv preprint arXiv:2309.13037*, 2023.

[126] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4583–4590. IEEE, 2021.

[127] 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.

[128] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020.

[129] Ted Xiao, Harris Chan, Pierre Sermanet, Ayzaan Wahid, Anthony Brohan, Karol Hausman, Sergey Levine, and Jonathan Tompson. Robotic skill acquistion via instruction augmentation with vision-language models. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[130] Annie Xie, Lisa Lee, Ted Xiao, and Chelsea Finn. Decomposing the generalization gap in imitation learning for visual robotic manipulation. *arXiv preprint arXiv:2307.03659*, 2023.

[131] Naoki Yokoyama, Sehoon Ha, and Dhruv Batra. Success weighted by completion time: A dynamics-aware evaluation criteria for embodied navigation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1562–1569. IEEE, 2021.

[132] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved VQGAN. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2022.

[133] 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.

[134] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Dee M, Jodilyn Peralta, Brian Ichter, Karol Hausman, and Fei Xia. Scaling robot learning with semantically imagined experience. In *arXiv preprint arXiv:2302.11550*, 2023.

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

[136] Xiaoshuai Zhang, Rui Chen, Ang Li, Fanbo Xiang, Yuzhe Qin, Jiayuan Gu, Z. Ling, Minghua Liu, Peiyu Zeng, Songfang Han, Zhiao Huang, Tongzhou Mu, Jing Xu, and Hao Su. Close the optical sensing domain gap by physics-grounded active stereo sensor simulation. *IEEE Transactions on Robotics*, 39:2429–2447, 2022.

[137] Yunchu Zhang, Liyiming Ke, Abhay Deshpande, Abhishek Gupta, and Siddhartha Srinivasa. Cherry-picking with reinforcement learning. *arXiv preprint arXiv:2303.05508*, 15, 2023.

[138] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems*, 2023.

[139] Chengmin Zhou, Bingding Huang, and Pasi Fränti. A review of motion planning algorithms for intelligent robots. *Journal of Intelligent Manufacturing*, 33(2):387–424, 2022.

[140] 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.