A Generative Paradigm for Building Generalist Robots: *Infrastructure*, *Scaling*, and *Policy Learning*

Xian Zhou

CMU-RI-TR-24-74

November, 2024

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:

Katerina Fragkiadaki, *chair*
David Held
Deepak Pathak
Chuang Gan, *UMass Amherst*
Shuran Song, *Stanford University*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

To my family,
without whom I wouldn't be able to begin this journey,

and to Tian Jiangshan,
without whom I wouldn't have the courage to complete it.

Abstract

Robotics researchers have been attempting to extend data-driven breakthroughs in fields like computer vision and language processing into robot learning. However, unlike vision or language domains where massive amounts of data is readily available on the internet, training robotic policies relies on physical and interactive data collected via interacting with the physical world – a resource-intensive process limited by labor constraints. Such data scarcity has long been a major bottleneck in scaling up robot learning systems, constraining prior efforts to small-scale and task-specific settings.

In this thesis, we present a generative paradigm that could potentially lead to general purpose robots by addressing existing limitations. This is achieved through three self-contained yet interdependent lines of work, which, when integrated, collectively form a cohesive and comprehensive paradigm:

- We propose building comprehensive world simulator infrastructures for modeling the physical world, both learning based and rule-based, to create a virtual yet realistic and powerful world for robotic agents to explore and develop their skills.

- We present *Generative Simulation*, a generative framework for autonomously scaling up robotic data generation better leveraging the power of compute, built on top of the world models we built. Traditional policy training in simulation has long been hindered by extensive human effort in designing tasks, assets, environments, training supervisions, and evaluation metrics. We design a robotic agent that automates all stages of simulated robot learning – from initial task proposal to policy training – leading to diverse robotic demonstrations.

- We present neural network architectures and learning methods for distilling collected demonstration data into unified multi-modal robotic policies, completing the cycle from data generation to effective policy training.

Acknowledgments

The past seven years has been an unconventionally long and challenging journey, marked by numerous ups and downs, including changing my advisor, being stranded abroad due to COVID, taking a gap year, pivoting my thesis topic, and leading an unprecedentedly big collaboration effort with members from all over the world to work together on an ambitious mission. As I finally approach its completion, I am deeply grateful for all that has been achieved along the way and for the impact it has made and will make on the relevant fields. Standing at this moment, I feel profoundly thankful to every person who offered guidance, support, encouragement, and inspiration during times when I faced various challenges.

First and foremost, I extend my deepest gratitude to my advisor, Katerina Fragkiadaki. Your wholehearted support—not only in research but also through various forms of care and mental encouragement—has been one of the most significant forces sustaining me throughout this journey. I am incredibly proud of how far we have come together and how we have both grown, becoming better as a researcher and an advisor. Our numerous discussions, though not always peaceful :), have undoubtedly led to better outcomes, shaping me into a more mature and capable researcher. These experiences have left an indelible mark, providing unforgettable memories and a source of strength for the journey ahead.

My sincere thanks go to David Held, Deepak Pathak, Chuang Gan, and Shuran Song for serving on my thesis committee. You are among the most influential researchers in robotics, and your invaluable feedback and support throughout my research journey have been instrumental in shaping the direction of my work and bringing this thesis to fruition. I am especially grateful to David Held and Shuran Song for their valuable suggestions and discussions when I first conceived the idea of generative simulation. Your guidance during my follow-up research work and other efforts such as organizing relevant conference workshops, have been incredibly helpful. Last but not least, I owe tremendous gratitude to Chuang Gan, who has been both an insightful research collaborator and a supportive mentor. Our collaboration began during my visit to Boston two years ago, and since then, we have worked together on many exciting and impactful research projects. This period has been transformative, allowing me to grow into an independent and mature researcher and to develop a comprehensive vision for a series of work. Many of the results and achievements that have significantly contributed to this thesis would not have been possible

without your support.

I am deeply grateful for the opportunity to collaborate with Yiling Qiao, Tsun-Hsuan Wang, and Zhenjia Xu over the past two years. Together, we formed an incredible team and led what is arguably one of the most ambitious projects in academia in recent years. Through our combined efforts, we built a transformative infrastructure that I believe will have a lasting impact on the robotics community in the years to come. I also want to express my gratitude to all the amazing collaborators and advisors who contributed to the monumental Genesis collaboration effort: Zhehuan Chen, Juntian Zheng, Ziyan Xiong, Yian Wang, Yunsheng Tian, Mingrui Zhang, Byungchul Kim, Pingchuan Ma, Tairan He, Zhiyang Dou, Xiaowen Qiu, Tianyu Li, Zhanlue Yang, Feng Chen, Zipeng Fu, Zilin Si, Zackory Erickson, Dan Gurfreund, Daniela Rus, Ming Lin, Guanya Shi, Bo Zhu, and many more. Time and again, I have been amazed by what we have achieved together, and I am proud to have been part of such an exceptional team.

I extend my heartfelt thanks to Yufei Wang, my best friend at CMU, for countless fruitful discussions—both research and non-research—and for the joy and camaraderie we shared. Your unwavering support has been invaluable during some of the most challenging times. I thank Hsiao-Yu Tung for all the unforgettable moments we shared in the lab at the second level in GHC, from coding up bugs together to making breakfasts and dumplings, and for having deep talks about life and dreams (copied from your thesis acknowledgment). I am deeply grateful to all the talented students and lab mates I have had the privilege to meet over the years at CMU: Max Sieb, Chengqian Che, Xiaofang Wang, Ye Yuan, Adam Harley, Theo Gervet, Yunchu Zhang, Shamit Lal, Gaurav Pathak, Wen-Hsuan Chu, Mihir Prabhudesai, Nikos Gkanatsios, Gabriel Sarch, Tsung-Wei Ke, Lei Ke and Kashu Yamazaki. Thank you for all the insightful discussions and collaborations throughout the years, which have enriched my academic journey.

I give special thanks to my friends that have been supporting me since my pre-PhD days: Xie Xian, Yu Hanxiao, Chen Ziwen, Cui Xiaoyang, Guo Pengsheng, Song Yixuan, and Wang Siqi. Your lasting support has been a guiding light, continually sustaining and encouraging me throughout this journey. I would also like to express deep gratitude to my undergraduate advisors: Pham Quang Cuong, I-Ming Chen, Ang Wei Tech, and Ooi Kim Tiow. This journey would never have begun without your support and

guidance.

Finally, I dedicate this thesis to Tian Jiangshan, without whom the past years would have been unimaginably difficult; to Milky and Nono — the former, a faithful companion who lay beside my keyboard every 5 a.m. I was up and still coding, and the latter, who showed incredible toughness today while undergoing a painful surgery for his tail and will always remain a cherished part of my life; to my parents, for giving me life and providing unconditional love and support ever since; and to my grandparents, whom I deeply wish could witness this moment with me — I believe they are.

I am proud of you all.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Motivated by recent data-driven successes made in relevant fields, from computer vision to natural language understanding and generation, robotics researchers have been attempting to extend data-driven approaches into robot learning to build general purpose robots that can master a diverse range of skills and operate in various non-factory settings. However, unlike vision or language domains where massive amounts of data is readily available on the internet, training robotic policies via a data-driven manner requires collecting physical and interactive data by interacting with the physical world – a resource-intensive process limited by labor constraints. Such data scarcity has long been a major bottleneck in scaling up robot learning systems, constraining prior efforts to small-scale and task-specific settings.

Efforts in scaling up robotic data collection and policy learning so far have been dominated by two pathways. The first pathway is to collect human-provided demonstration data in real-world, via either kinesthetic teaching or teleoperation [21, 36, 45, 177], or leveraging visual human demonstration [253, 282, 319]. However, this approach struggles with generating wide distribution demonstration data, and is intrinsically difficult to scale up due to reliance on human labor. Another promising approach is to train robotics skills in simulation, which naturally enjoys the power of compute and ideally, can scale effortlessly by leveraging ever-increasing computation power. However, training robotic skills in simulation have also been hindered by issues from various aspects: efficiency and capability of simulation platforms limit the possible interactions carried out by robots in simulated worlds; in addition, there

exist multiple stages before launching the actual skill training in this pipeline that still requires tremendous human input.

In this thesis, we present a comprehensive approach that could potentially tackle the data scarcity problem in robotics, and propose a set of methods that each tackles an individual component of the compete pipeline, completing the whole cycle from data generation to policy training. Our idea is simple: by leveraging the latest foundation and generative models from other useful modalities, we propose to auto-generate all the static and semantic information needed for constructing training environments and supervisions for learning robotic skills: from task proposal, to 3D assets, interactive environments, training supervisions such as reward functions, to close-loop evaluations of trained skills. After that, we construct simulation environments and training pipelines, and then resort to either reinforcement learning or trajectory optimization to automatically train the robots, and thus yielding desired demonstrations for the proposed tasks. This paradigm, we call *Generative Simulation*, requires methodological improvements in multiple aspects: first of all, we need to design a robust and self-sustained framework that is able to execute the whole pipeline autonomously; second, training a wide distribution of proposed skills requires inherent support from the simulation infrastructure and physics engine itself; third, we need to devise unified and effective policy architectures that we can distill the generated demonstration into via imitation learning. As a result, this thesis consists of three parts:

- **Building Structured and Universal World Simulators**: In this part, we discuss our effort in constructing efficient and effective simulation infrastructures, both learning-based and rule-based, for providing a virtual ground where robots can explore and train skills better leveraging compute.
- **Generative Simulation**: We then discuss in detail the whole proposed pipeline for scaling up robot learning, by auto-generating robotic demonstration data via generative simulation.
- **Policy Architectures**: In the last part, we device novel neural architectures that empowers robotic policy that can be trained via imitating the generated demonstration data. Our proposed policy takes as input both visual observations and natural language instructions, and produces physical actions for completing

designated tasks.

1.1 Building Structured and Universal World Simulators

Humans can effortlessly imagine how their surrounding environments will change as a result of their actions. Such dynamics model we have allows us to perform mental simulation of the world and plan our actions accordingly. We point out two important aspects of such mental simulation models that are important for their generalization: 1) their reasoning is carried out in a 3D spatial representation, not affected by variations in view point and any occlusions; 2) we observe how the environment behaves in response to our actions and quickly adapt these model for the situation at hand based on new observations. Motivated by these, we propose two important improvements towards learning a more generalizable dynamics model. First, we argue that scene dynamics are simpler to learn and represent in 3D than in 2D. In 3D, object appearance and object location are disentangled. Dynamics prediction by moving around objects is not possible in a projective 2D image space, since objects change appearance due to camera viewpoint variation, occlusions or out-of-plane object rotations. In addition, inferring free space and object collisions is natural in a 3D representation space. To this end, we propose 3D-OES in Chapter 2, an action-conditioned dynamics model that predicts scene changes caused by object and agent interactions in a viewpoint-invariant 3D neural scene representation space, inferred from RGB-D videos. Second, we argue that it is important for a robot to infer environment properties from its interaction with the surroundings, and adapt its dynamics models accordingly. In Chapter 3, we propose HyperDynamics, a dynamics meta-learning framework that generates parameters for dynamics models dedicated to the situation at hand, based on observations of how the environment behaves.

Learning-based dynamics models could be useful for model-based planning, but to generate their training data, we still need physic-based simulation infrastructures. The advancement in physics-based simulation has become a crucial driving force in both dynamics model learning and robotic skill training. However, most previous work in robotic manipulation and most existing simulation platforms still mainly

consider relatively simplistic rigid object settings. Recently, there has been a rising interest in studying deformable objects. In this thesis, we would like to build simulation infrastructure with improvements on multiple aspects. First of all, we would like to extend to more complex environment setting where the robots need to interact with various non-rigid objects, from deformable objects, to fluids and granular materials. We first present FluidEngine and FluidLab in Chapter 4. The former is a fully-differentiable general-purpose physics engine that supports a wide variety of materials and their couplings, providing capability for simulating complex fluid manipulation tasks. The latter is a suite of standardized evaluation tasks built around the former, that we proposed to evaluate various methods in the context of fluid manipulation. We empirically show that our proposed optimization scheme, when coupled with our fully differentiable simulation model, is able to address the proposed tasks well, outperforming gradient-free methods in terms of both sample efficiency and final performance. Then, we go beyond this domain-specific simulation engine, and introduce Genesis, a large-scale collaborative effort for building a universal, high-performance, and easy-to-use simulation infrastructure for robotics researchers. Genesis integrated various state-of-the-art physics solvers into a unified framework, designed a fully differentiable architecture, and supports more advanced tactile sensing module and real-time photorealistic rendering system.

1.2 Automating Robotic Demonstration Data Generation via Generative Simulation

Generalist robot manipulators need to learn a wide variety of manipulation skills across diverse environments. Current robot training pipelines rely on humans to provide kinesthetic demonstrations or to program simulation environments and to code up reward functions for reinforcement learning. Such human involvement is an important bottleneck towards scaling up robot learning across diverse tasks and environments. In this part of the thesis, we propose a systematic pipeline that leverages generative models with multiple modalities, and combine them with powerful simulation infrastructures, to automate the whole pipeline for robotic policy training in simulation. Specifically, our system aims to equip an intelligent robotic agent

with the capability for first proposing useful skills on its own, and then construct a semantically meaningful environments with relevant assets to populate in the environment. Afterwards, the agent leverages large language models to compose reward functions that can then be used for skill training. Finally, the agent launches the training in the constructed environment, and continuously produce trained demonstration data associated with the proposed task settings.

Specifically, we first present Gen2Sim (Chapter 6), a initial proof of concept of this proposed idea. We generate 3D assets for simulation by lifting open-world 2D object-centric images to 3D using image diffusion models and querying LLMs to determine plausible physics parameters. Given URDF files of generated and human-developed assets, we chain-of-thought prompt LLMs to map these to relevant task descriptions, temporal decompositions, and corresponding python reward functions for reinforcement learning. We show Gen2Sim succeeds in learning policies for diverse long horizon tasks, where reinforcement learning with non temporally decomposed reward functions fails. Gen2Sim provides a viable path for scaling up reinforcement learning for robot manipulators in simulation, both by diversifying and expanding task and environment development, and by facilitating the discovery of reinforcementlearned behaviors through temporal task decomposition in RL. Our work contributes hundreds of simulated assets, tasks and demonstrations, taking a step towards fully autonomous robotic manipulation skill acquisition in simulation. Then, we present RoboGen (Chapter 7), a more comprehensive and self-contained system that extends to deformable objects manipulation, with improved environment generation. Robogen agent first proposes interesting tasks and skills to develop, and then generates simulation environments by populating pertinent assets with proper spatial configurations. Afterwards, the agent decomposes the proposed task into sub-tasks, selects the optimal learning approach (reinforcement learning, motion planning, or trajectory optimization), generates required training supervision, and then learns policies to acquire the proposed skill. RoboGen can be queried repeatedly, producing an endless stream of skill demonstrations associated with diverse tasks and environments. We empirically demonstrate that tasks and demonstrations generated by RoboGen is more diverse than prior human-crafted robotic datasets.

1.3 Structured and Unified Neural Architectures for Robotic Policy Learning

Data driven robotic policy learning requires good policy architecture with appropriate inductive biases to imitate collected demonstrations. In the last part of the thesis, we present a series of work that aims to build a performance neural architecture to empower a multi-modal multi-task robotic policy. Our insights for improving existing policy architectures are two folds. First of all, we argue that 3D perceptual representations are well suited for robot manipulation as they easily encode occlusions and simplify spatial reasoning. Many manipulation tasks require high spatial precision in end-effector pose prediction, which typically demands high-resolution 3D feature grids that are computationally expensive to process. As a result, most manipulation policies operate directly in 2D, foregoing 3D inductive biases. In Chapter 8, we introduce Act3D, a manipulation policy transformer that represents the robot's workspace using a 3D feature field with adaptive resolutions dependent on the task at hand. The model lifts 2D pre-trained features to 3D using sensed depth, and attends to them to com- pute features for sampled 3D points. It samples 3D point grids in a coarse to fine manner, featurizes them using relative-position attention, and selects where to focus the next round of point sampling. In this way, it efficiently computes 3D action maps of high spatial resolution. Act3D sets a new state-of-the-art in RLBench, an established manipulation benchmark, where it achieves 10% absolute improvement over the previous SOTA 2D multi-view policy on 74 RLBench tasks and 22% absolute improvement with 3x less compute over the previous SOTA 3D policy. In Chapter **??**, we present ChainedDiffuser, a policy architecture that further unifies action keypose prediction and trajectory diffusion generation for learning robot manipulation from demonstrations. The main innovation is to use a global action predictor to predict actions at keyframes, a task that requires multi-modal semantic scene understanding, and to use a local trajectory diffuser to predict trajectory segments that connect predicted macro-actions. ChainedDiffuser outperforms both state-of-the-art keypose (macro-action) prediction models (including Act3D) that use motion planners for trajectory prediction, and trajectory diffusion policies that do not predict keyframe macro-actions. We conduct experiments in both simulated and

real-world environments and demonstrate ChainedDiffuser's ability to solve a wide range of manipulation tasks involving interactions with diverse objects.

1.4 Thesis Structure

The thesis is split into three parts, corresponding to the three threads of works described above. The main chapters are mostly standalone published papers, and we provide the references below to also serve as an outline of the thesis:

1. **Building Structured and Universal World Simulators**
    - Chapter 2: Viewpoint-Invariant Object-Factorized Environment Simulators [278] (CoRL 2020)
    - Chapter 3: Meta-learning Object and Agent Dynamics with Hypernetworks [301] (ICLR 2021)
    - Chapter 4: A Differentiable Environment for Benchmarking Complex Fluid Manipulation [305] (ICLR 2023)
    - Chapter 5: A Universal and Generative Physics Engine for Robotics and Beyond (on-going work)
2. **Automating Robotic Demonstration Data Generation via Generative Simulation**
    - Chapter 6: Scaling Up Robot Learning in Simulation with Generative Models [278] (ICRA 2024)
    - Chapter 3: RoboGen: Towards Unleashing Infinite Data for Automated Robot Learning via Generative Simulation [291] (ICLR 2024)
3. **Unified Neural Architectures for Multi-modal Multi-task Policy Learning**
    - Chapter 8: 3D Feature Field Transformers for Multi-task Robotic Manipulation [76] (CoRL 2023)
    - Chapter **??**: Unifying Trajectory Diffusion and Keypose Prediction for Robotic Manipulation [304] (CoRL 2023)

Finally, in Chapter 10, we present a summary to conclude the presented work,

and discuss on-going and future work along the proposed thread.

Part I

Building Structured World Simulators for Modeling the Physical World

Chapter 2

Learning Viewpoint-Invariant 3D Dynamics Models

2.1 Introduction

Humans can effortlessly imagine how a scene will change as a result of their interactions with the objects in the scene [49, 69]. What is the representation space of these imaginations? They are not pixel accurate and, interestingly, they are not affected by occlusions. Consider a teaspoon dipping inside a coffee mug. Though it will be occluded from nearly all viewpoints but the bird's eye view, we have no difficulty keeping it in our mind as present and complete. We can imagine watching it from different viewpoints, increase or decrease its size, predict whether it will fit inside the mug, or even imagine filling the mug with more spoons.

Inspired by human's capability to simulate scene changes in a viewpoint-invariant and occlusion-resistant manner, we present 3D object-factorized environment simulators (3D-OES), an action-conditioned dynamics model that predicts scene changes caused by object and agent interactions in a viewpoint-invariant 3D neural scene representation space, inferred from RGB-D videos. 3D-OES differentiably maps an RGB-D image to a 3D neural scene representation, detects objects in it, and forecasts their future 3D motions, conditioned on actions of the agent. A graph neural network operates on the extracted 3D object feature maps and the action input and

predicts object 3D translations and rotations. Our model then generates future 3D scenes by simply translating and rotating object 3D feature maps, inferred from the *first time step*, according to cumulative 3D motion predictions. In this way, we avoid distribution shift in object features caused by forward model unrolling, hence minimizing error accumulation.

Our main insight is that scene dynamics are simpler to learn and represent in 3D than in 2D, for the following reasons: i) **In 3D, object appearance and object location are disentangled.** This means object appearance (what) does not vary with object locations (where). This what-where disentanglement permits generating scene variations by simply translating and rotating 3D object appearance representations. Scene generation by moving around objects is not possible in a projective 2D image space, since objects change appearance due to camera viewpoint variation, occlusions or out-of-plane object rotations [58]. It is precisely the permanence of object appearance in 3D that permits easy simulation. ii) **In 3D, inferring free space and object collisions is easy.** Given a 3D scene description in terms of object locations and 3D shapes, we can easily predict whether an object will collide with another or will be contained in another. Similar inferences would require many examples to learn directly from 2D images, and would likely have poor generalization. Yet, extracting 3D scene representations from RGB or RGB-D video streams is a challenging open problem in computer vision research [110, 134, 226, 227, 276]. We build upon the recently proposed geometry-aware recurrent neural networks (GRNNs) [91, 276] to infer 3D scene feature maps from RGB-D images in a differentiable manner, optimized end-to-end for our object dynamics prediction task.

We evaluate 3D-OES in single-step and multi-step object motion prediction for object pushing and falling, and apply it for planning to push objects to desired locations. We test its generalization while varying the number and appearance of objects in the scene, and the camera viewpoint. We compare against existing learning-based 2D image-centric or object-centric models of dynamics [87, 317] as well as graph-based dynamics learned over engineered 3D representations of object locations [232]. Our model outperforms them by a large margin. In addition, we empirically show that training the 2D baselines under varying viewpoints causes them to dramatically underfit on the training data, and be highly inaccurate in the validation set. This suggests that different architectures are necessary to handle

viewpoint variations in dynamics learning and 3D-OES is one step in that direction.

In summary, the main contribution of this work is a graph neural network over 3D object feature maps extracted from convolutional end-to-end differentiable 3D neural scene representations for forecasting 3D object motion. Graph networks are widely used in 2D object motion interaction predictions [13, 116, 130, 317]. We show that by porting such relational reasoning in an 3D object-factorized space, object motion prediction can generalize across camera viewpoints, lifting a major limitation of previous works on 2D object dynamics. Moreover, future and counterfactual scenes can be easily generated by translating and rotating 3D object feature representations. In comparison to recent 3D particle graph networks [146, 188, 234], our work can operate over input RGB-D images and not ground-truth particle graphs. In comparison to recent scene-specific image-to-3D particle graph encoders [147], our image to 3D scene encoder can generalize across environments with novel objects, novel number of objects, and novel camera viewpoints, Moreover, our model presents effective sim-to-real transfer to a real-world robotic setup.

2.2 Related Work

The inability of systems of physics equations to capture the complexity of the world [295] has led many researchers to pursue learning-based models of dynamics, or combine those with analytic physics models to help fight the undermodeling and uncertainty of the world [4, 325]. Learning world models is both useful for model-based control [63, 82, 87] as well as a premise towards unsupervised learning of visuomotor representations [1, 212]. Several formulations have been proposed, under various names, such as world models [82], action-conditioned video prediction [199], forward models [183], neural physics [29], neural simulators, etc. A central question is the representation space in which predictions are carried out. We identify two main research threads:

(i) Methods that **predict the future in a 2D projective space**, such as future visual frames [60, 64, 178, 198], neural encodings of future frames [1, 37, 82, 87, 207], object 2D motion trajectories [13, 29, 67], 2D pixel motion fields [25, 58, 63, 64]. Models that predict motion and use it to warp pixel colors forward in time need to handle occlusions as well as changes in size and aspect ratio. Though increasing

the temporal memory from where to borrow pixel colors helps [58, 136], the model needs to learn many complex relationships between pixels to handle such changes over time. Recent architectures incorporate object-centric and relational biases in order to generalize across varying number of objects present in the image, using graph neural networks [13, 14, 116, 130, 317]. Cross-object interactions are captured using edges in an entity graph, where messages are iteratively exchanged between the nodes to update each other's embedding [13, 232]. Most works that use object factorization biases either assume the objects' segmentation masks are given or are trivial to obtain from color segmentation [13, 29, 67]. These models work well under a static and fixed camera across training and test conditions, but cannot effectively generalize across camera viewpoints [47] as we empirically validate.

(ii) methods that **predict the future in a 3D space** of object or particle 3D locations and poses extracted from the RGB images using human annotations [8] or assumed given [145, 187, 296]. Such explicit 3D state representations are hard in general to obtain from raw RGB input in-the-wild, outside multiview environments [8, 145]. The work of [147] attempts to extract the particle 3D locations directly from images, however the encoder proposed is specific to the scene. Different encoders are trained for different scene image to particle mappings.

Our work builds upon learnable 2D-to-3D convolutional encoders that extract 3D scene representations from images and are trained for self-supervised view prediction, along with tasks of 3D object detection and 3D motion forecasting, relevant for 3D object dynamics learning. Our image-to-3D scene encoders and object detectors generalize across object appearance and number of objects, and do not assume any ground-truth information about the object or particle locations [146, 188, 234] or image segmentation [297] during test time.

2.3 Object-Factorized Environment Simulators (3D-OES)

The architecture of 3D-OES is depicted in Figure 2.1. At each time step, our model takes as input a single or a set of RGB-D images of the scene along with the corresponding camera views to capture them, and encodes these inputs into a

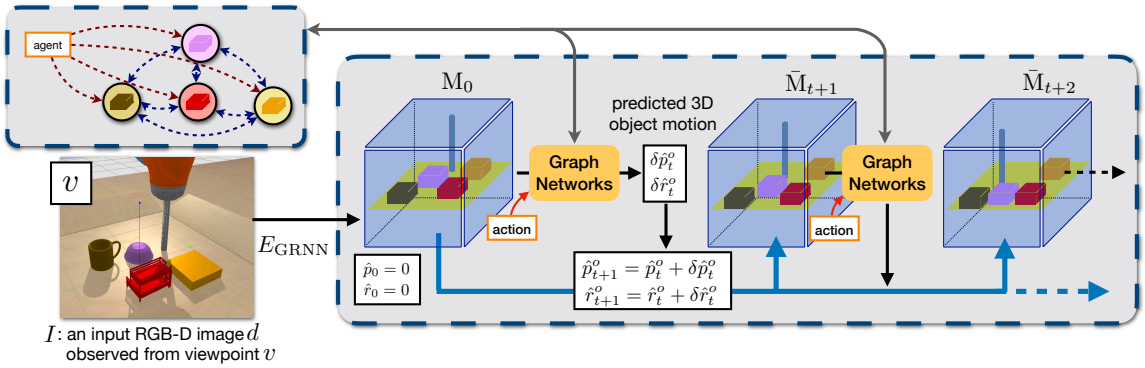


Figure 2.1: **3D-OES** predict 3D object motion under agent-object and object-object interactions, using a graph neural network over 3D feature maps of detected objects. Node features capture the appearance of an object node and its immediate context, and edge features capture relative 3D locations between two nodes, so the model is translational invariant. After message passing between nodes, the node and edge features are decoded to future 3D rotations and translations for each object.

3D scene feature representation by neurally mapping image and depth maps to 3D feature grids (Section 2.3.1). Then, it detects 3D object boxes in the inferred 3D scene representation, and crops the scene representation to obtain a set of object-centered 3D feature maps. A graph neural network over the object nodes will take as inputs object appearances and the agent actions and predict the future 3D rotation and translation for each object (Section 2.3.2). We will assume for now rigid objects, and we discuss in Section 8.7 how to extend our framework to deformable and articulated objects. Our model generates future scenes by warping object-centric 3D feature maps with the predicted cumulative 3D object motion (Section 2.3.2). These synthesized future 3D scene feature maps, though not directly interpretable, can be decoded to RGB images from any desired camera viewpoints via a neural renderer to aid interpretability. We use long-term simulations of 3D-OES to generate action plans for pushing objects to desired locations in cluttered environments using model predictive control (Section 2.3.3). We apply our model to learn dynamics of objects pushed around on a table surface and objects falling on top of others. At training time, we assume access to 3D object bounding boxes to train our 3D object detector.

2.3.1 Differentiable 2D-to-3D lifting with Geometry-Aware Recurrent Networks (GRNNs)

Geometry-Aware Recurrent Networks (GRNNs) introduced in [91, 276] are network architectures equipped with a differentiable unprojection (2D-to-3D) module and a 3D scene neural map as their bottleneck. They can be trained end-to-end for a downstream task, such as supervised 3D object detection or self-supervised view prediction. We will denote the 3D scene feature as $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$ where w, h, d, c denote width, height, depth and number of channels, respectively. Every (x, y, z) grid location in the 3D feature map $\mathbf{M}$ holds a c-dimensional feature vector that describes the semantic and geometric properties of a corresponding physical location in the 3D world scene. Given an input video, GRNNs estimate the relative camera poses between frames, and transform the inferred 3D features map $\mathbf{M}_t$ to a world coordinate frame to cancel the camera egomotion, before accumulating it with 3D feature maps across time steps. In this way, information from 2D pixels that correspond to the same 3D physical point end up nearby in the 3D neural map. We use such cross-view registration in case we have access to concurrent multiple camera views for the first timestep of our simulations. Upon training, GRNNs map RGB-D images or a single RGB-D image to a *complete* 3D feature map of the scene they depict, i.e., the model learns to *imagine* the missing or occluded information from the input view. We denote this 2D-to-3D mapping as $\mathbf{M} = E_{\text{GRNN}}(I_1, ..., I_t)$, where $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$ and $I_t = \{d_t, v_t\}$ denotes the RGB-D image d_t and the corresponding camera pose v_t at time step t. Note that the input can be a single RGB-D view, in which case $\mathbf{M} = E_{\text{GRNN}}(I)$. For further details on GRNNs, please refer to [91, 276].

View prediction for visualizing latent 3D neural simulations We train GRNNs end-to-end for RGB view regression in videos of static scenes and moving cameras as proposed in [276], by neurally projecting the 3D scene feature maps and mapping them to 2D images. Our decoder involves a differentiable 3D-to-2D projection module that projects the 3D scene feature representation after orienting it to the query camera viewpoint. The projected features are then decoded into images through a learned decoder. In this way, the trained projection and decoding module can be used to interpret and visualize the 3D latent feature space with view-specific 2D images, given any desired camera viewpoint.

3D object detection Our model uses a 3D object detector to map the 3D scene neural map $\mathbf{M}$ to a variable number of object axis-aligned 3D boxes and corresponding 3D segmentation masks, i.e., binary 3D voxel occupancies: $\mathcal{O} = Det(\mathbf{M}), \mathcal{O} = \{\hat{b}^o = (p_x^o, p_y^o, p_z^o, w^o, h^o, d^o) \in \mathbb{R}^6, m^o \in \{0,1\}^{w^o \times h^o \times d^o}, o = 1 \cdots |\mathcal{O}|\}$, where p_x^o, p_y^o, p_z^o stands for the 3D box centroid and w^o, h^o, d^o stands for 3d box size. Its architecture is similar to Mask R-CNN [93] but uses 3D input and output instead of 2D. Given an object 3D centroid p_x^o, p_y^o, p_z^o, we crop the 3D scene feature map $\mathbf{M}$ using a corresponding fixed-size axis-aligned 3D bounding box to obtain corresponding object-centric feature maps $\mathbf{M}^o, o = 1 \cdots |\mathcal{O}|$ for all objects in the scene.

2.3.2 3D Object Graph Neural Networks for Motion Forecasting

Objects are the recipient of forces exercised by active agents; meanwhile, objects themselves carry momentum and cause other objects to move. How can we model cross-object dynamic relationships in a way that generalizes with varying number of objects and arbitrary chains of interactions?

We consider a graph interaction network [13] over the graph comprised of the detected objects and the agent's end-effector. Inputs to the network are the object-centric feature maps, one per object node, the objects' velocities, the agent's action represented as a 3D translation, as well as edge features, which incorporate the relative 3D displacements between the nodes. The outputs of the network are the 3D translations $\delta\hat{p}$ and 3D relative rotations $\delta\hat{r}$ of the object nodes at the next time step. During message passing in the constructed graph, edge and node features are encoded and concatenated, and messages from neighboring nodes are aggregated via summation. Our graph network is trained supervised to minimize a standard regression loss for the next time step.

Forward unrolling with object appearance permanence To predict long term results of actions, as well as results of action sequences, the model needs to be unrolled forward in time as commonly done in related works [13, 14, 116, 130, 317]. Different from previous works though, 3D-OES can synthesize 3D neural scenes of future timesteps by warping (translating and rotating) object feature maps obtained from the first timestep—as opposed to the ones obtained from the predicted scene of

the previous timestep—according to cumulative 3D motion predictions. Specifically, given predicted 3D object motions ($\delta\hat{p}_t$, $\delta\hat{r}_t$) at an unrolling step t, we estimate the **cumulative** 3D rotation and translation of the object with respect to the first timestep:

$$\hat{p}_t = \hat{p}_{t-1} + \delta\hat{p}_t, \quad \hat{r}_t = \hat{r}_{t-1} + \delta\hat{r}_t, \quad t = 1 \cdots T, \quad \hat{p}_0 = 0 \quad \hat{r}_0 = 0. \tag{2.1}$$

where T denotes the number of unrolling steps thus far. Then, given 3D object segmentation masks m^o and object-centric 3D feature maps $\mathbf{M}^o$ obtained by the 3D object detector from the input RGB-D image, we rotate and translate the object masks and 3D feature maps using the cumulative 3D rotation $\hat{r}_t$ and 3D translation $\hat{p}_t$ using 3D spatial transformers. We synthesize a new 3D scene feature map $\bar{\mathbf{M}}_t$ by placing each transformed object-centric 3D feature map at its predicted 3D location: $\bar{\mathbf{M}}_t = \sum_{o=1}^{|\mathcal{O}|} \text{Draw}(\text{Rotate}(m^o, \hat{r}_t^o) \odot \text{Rotate}(\mathbf{M}^o, \hat{r}_t^o), \hat{p}_t^o)$, where superscript o denotes the object identity, $\text{Rotate}(\cdot, r)$ denotes 3D rotation by angle r, $\odot$ denotes voxel-wise multiplication, and $\text{Draw}(\mathbf{M}, p)$ denotes adding a feature tensor $\mathbf{M}$ at a 3D location p. This synthesized scene map is used for neural rendering to help interpret the predicted scene at t. To obtain the inputs for our graph neural network at the next time step, we can potentially crop the synthesized 3D scene map $\bar{\mathbf{M}}_t$ at the predicted 3D location. However, we find that directly using object features obtained in the first time step and including accumulative relative object pose as part of the object state works better in practice. Our graph neural motion forecaster is trained through forward unrolling. Error of each time step is back-propagated through time. More implementation details are included in Appendix Section C.1.

2.3.3 Model Predictive Control with 3D-OES

Action-conditioned dynamics models, such as 3D-OES, simulate the results of an agent's actions and permit successful control in zero-shot setups: achieving a specific goal in a novel scene without previous practice. We apply our model for pushing objects to desired locations in cluttered environments with model predictive control. Given an input RGB-D image I that contains multiple objects, a goal configuration is given in terms of the desired 3D location of an object $\mathbf{x}_{goal}^o$.

3D-OES infer the scene 3D feature map $\mathbf{M} = E_{\text{GRNN}}(I)$ and detects the objects

present in the scene. We then unroll the model forward in time using randomly sampled action sequences, as described in Section 2.3.2. We evaluate each action sequence based on the Euclidean distance from the goal to the predicted location $\hat{x}_T^o$ (after T time steps) for the designated object. We execute the first action of the best action sequence and repeat [269]. Our model combines 3D perception and planning using learned object dynamics in the inferred 3D scene feature map. While most previous works choose bird's eye viewpoints to minimize cross-object or robot-object occlusions [62], our control framework **can use any camera viewpoint**, thanks to its ability to map input 2.5D images to complete, viewpoint-invariant 3D scene feature maps. We empirically validate this claim in our experimental section.

2.4 Experiments

We evaluate our model on its prediction accuracy for single- and multi-step object motion forecasting under multi-object interactions, as well as on its performance in model predictive control for pushing objects to desired locations on a table surface in the presence of obstacles. We ablate generalization of our model under varying camera viewpoints and varying number of object and varying object appearance. Our model is trained to predict 3D object motion during robot **pushing** and **falling** in the Bullet Physics Simulator. For **pushing**, we have objects pushed by a Kuka robotic arm and record RGB-D video streams from multiple viewpoints. We create scenes using 31 different 3D object meshes, including 11 objects from the MIT Push dataset [320] and 20 objects randomly selected from *camera*, *mug*, *bowl*, and *bed* object categories of the ShapeNet dataset [28]. At training time, each scene contains at most *two* objects. We test with varying number of objects. For **falling**, we use 3D meshes of the objects introduced in [116], including a variety of shapes. We randomly select 1-3 objects and randomly place them on a table surface, and let one object fall from a height. We train our model with three camera views, and use either three or one randomly selected views as input during test time.

We compare 3D-OES against a set of baselines designed to cover representative models in the object dynamics literature: **(1)** *graph-XYZ*, a model that mimics Interaction Networks [13] and [8, 296]. It is a graph neural network in which object features are the 3D object centroid locations and their velocities, and edge features

are their relative 3D locations. **(2)** *graph-XYZ-image*, a model using graph neural network over 3D object centroid locations and object-centric 2D image CNN feature embeddings, similar to [317]. The model further combines camera pose information with the node features. **(3)** Visual Foresight (*VF*) [59], a model that uses the current frame and the action of the agent to predict future 2D frames by "moving pixels" based on predicted 2D pixel flow fields. **(4)** *PlaNet* [87], a model that learns a scene-level embedding by predicting future frames and the reward given the current frame.

We compare our model against baselines *graph-XYZ* and *graph-XYZ-image* on both motion forecasting and model predictive control. Since *VF* and *PlaNet* forecast 2D pixel motion and do not predict explicit 3D object motion, we compare against them on the pushing task with model predictive control. For further details on data collection, train-test data split, and implementation of the baselines, please refer to Appendix (Section B and C.2).

Table 2.1: **3D object motion prediction test error during object pushing in scenes with two objects** for 1,3, and 5 timestep prediction horizon.

Experiment Setting	Model		T=1	T=3	T=5
3 views (random, novel) + gt-bbox	*graph-XYZ* [13]	translation(mm)	4.6	32.1	66.3
		rotation(degree)	2.8	16.7	26.4
	graph-XYZ-image [317]	translation(mm)	6.0	39.3	69.7
		rotation(degree)	3.4	29.8	30.7
	Ours	translation(mm)	**3.6**	**22.5**	**43.4**
		rotation(degree)	**2.5**	**12.0**	**20.6**
1 view (random, novel) + gt-bbox	*graph-XYZ-image* [317]	translation(mm)	6.0	39.3	69.7
		rotation(degree)	3.4	29.8	30.7
	Ours	translation(mm)	**4.1**	**23.6**	**43.8**
		rotation(degree)	**3.1**	**12.2**	**20.3**
1 view (random, novel) + predicted-bbox	*graph-XYZ* [13]	translation(mm)	6.7	35.4	68.2
		rotation(degree)	3.0	20.1	30.32
	graph-XYZ-image [317]	translation(mm)	6.6	43.1	71.2
		rotation(degree)	3.6	31.8	32.4
	Ours	translation(mm)	**4.3**	**25.2**	**47.0**
		rotation(degree)	**2.7**	**12.1**	**19.7**
1 view (**fixed, same as train**) + predicted-bbox	*graph-XYZ-image* [317]	translation(mm)	5.1	29.6	54.5
		rotation(degree)	2.6	11.0	16.9

Table 2.2: **3D object motion prediction test error during object falling in scenes with three to four objects** for 1,3, and 5 timestep prediction horizon.

Experiment Setting	Model		T=1	T=3	T=5
1views (random, novel) + predicted-bbox	*graph-XYZ* [13]	translation(mm)	5.2	**11.7**	278.6
		rotation(degree)	**5.7**	**10.4**	43.28
	graph-XYZ-image [317]	translation(mm)	8.4	17.0	620.2
		rotation(degree)	9.2	16.6	117.9
	Ours	translation(mm)	**5.0**	13.1	**16.4**
		rotation(degree)	6.1	12.6	**18.7**

2.4.1 Action-Conditioned 3D Object Motion Forecasting

We evaluate the performance of our model and the baselines in single- and multi-step 3D motion forecasting for **pushing** and **falling** on novel objects in Tables 2.1 and 2.2 in terms of translation and rotation error. We evaluate the following ablations: i) using 1 or 3 camera views at the first time step, ii) using goundtruth 3D object boxes (gt-bbox) or 3D boxes predicted by our 3D detector, iii) varying camera viewpoints (random) versus keeping a single fixed camera viewpoint at train and test time. Our model outperforms the baselines both in translation and rotation prediction accuracy. When tested with object boxes predicted by the 3D object detector (Section 2.3.1) as opposed to ground-truth 3D boxes, our model is the least affected. *graph-XYZ-image* performs on par with or even worse than *graph-XYZ*, indicating that it does not gain from having access to additional appearance information. We hypothesize this is due to the way appearance and camera pose information are integrated in this baseline: the model simply treats camera pose information as additional input, as opposed to our model, which leverages geometry-aware representations that retain the geometric structure of the scene.

Multi-step forward unrolling The *graph-XYZ* baseline can be easily unrolled forward in time without much error accumulation since it does not use any appearance features. Still, as seen in Tables 2.1 and 2.2, our model outperforms it. *graph-XYZ* is oblivious to the appearance of the object and thus cannot effectively adapt its predictions to different object shapes.

Varying number of camera views Our model accepts a variable number of views as input, and improves when more views available; yet, it can accurately predict future motion even from a single RGB-D view. The prediction error of our

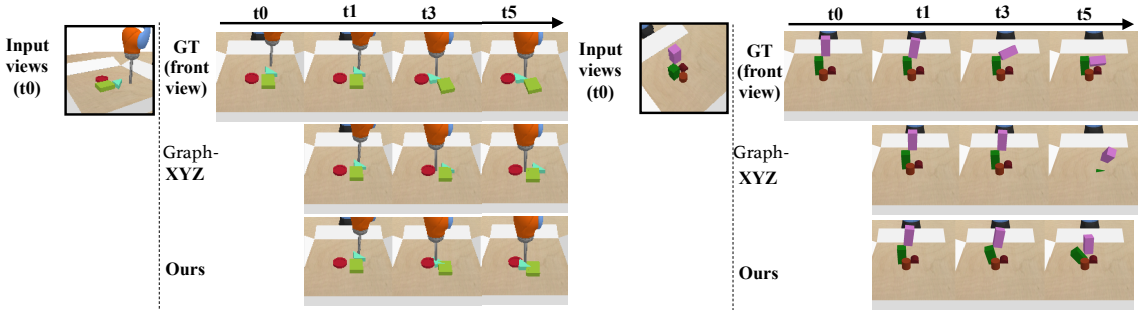


Figure 2.2: **Forward unrolling of our dynamics model and the *graph-XYZ* baseline.** Left: pushing. Right: falling. In the top row, we show (randomly sampled) camera views that we use as input to our model. The second row shows the ground-truth motion of the object from the front view. Rows 3, 4 show the predicted object motion from our model and the *graph-XYZ* baseline from the same front camera viewpoint. Our model better matches the ground-truth object motion than the *graph-XYZ* baseline. The latter does not capture object appearance in any way.

single view model is only slightly higher than the model using three random views as input. As shown in Table 2.1, the *graph-XYZ-image* baseline performs the worst and does not improve with more views are available. We believe this is due to the geometry-unaware way of combining multiview information by concatenation, though the model does have access to camera poses of the input images.

Varying camera viewpoint versus fixed camera viewpoint We show in Table 2.1 (last 2 rows) that *graph-XYZ-image* can achieve much better performance when trained and tested on a single fixed camera viewpoint. This is a setting widely used in the recently popular learning-based visual-motor control literature [59, 63, 208, 316], which restricts the corresponding models to work only under carefully controlled environments with a fixed camera viewpoint, while ours performs competitively to these model but also handles arbitrary camera viewpoints.

Visualization of the 3D motion predictions In Figure 2.2, we show qualitative long term motion prediction results produced by unrolling our model forward in time (more are shown in the supplementary video). Our model generalizes to novel objects and scenes with varying number of objects, though trained only on 2 object scenes.

Neural rendering and counterfactual simulations 3D-OES not only can simulate the future state of the scene, it also provides us a way to interpret the latent 3D representation and a space to run counterfactual experiements. We visualize the

latent 3D feature map by neurally projecting it from a camera viewpoint to an image through a learned neural decoder, and show the resulting images in Figure 2.3. We also show that our 3D representation allows us to alter the observed scene and run conterfactual simulations in multiple ways. More results are provided in Appendix Section A.

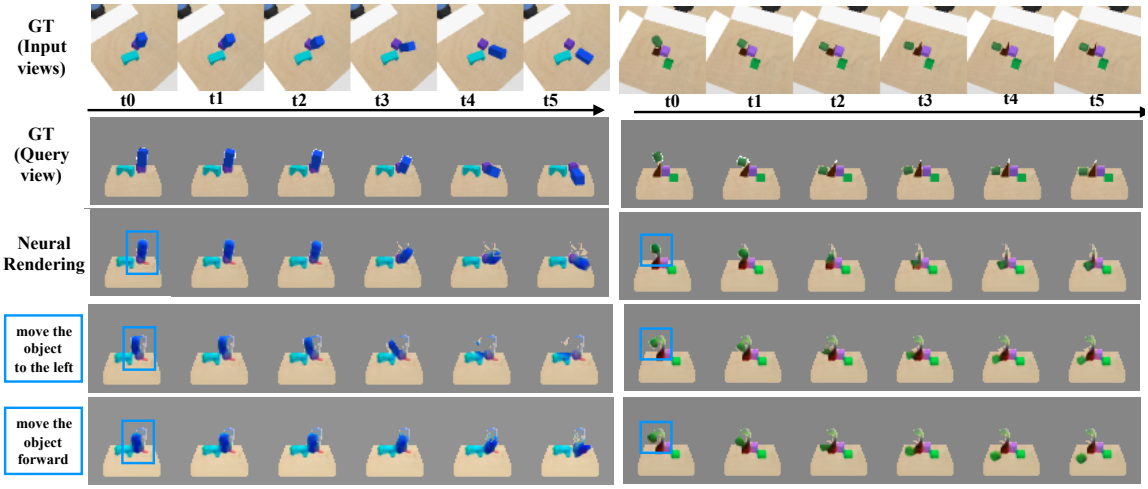


Figure 2.3: **Neurally rendered simulation videos of counterfactual experiments.** The first row shows the ground truth simulation video from the dataset. Only the first frame in this video is used as input to our model to produce the predicted simulations. The second row shows the ground truth simulation from a query view. The third row shows the future prediction from our model given the input image. The following rows show the simulation after manipulating an objects (in the blue box) according the instruction on the left most column.

2.4.2 Pushing with Model Predictive Control (MPC)

We test 3D-OES on pushing objects to desired locations using MPC and report the results in Table 2.3. For our model and *graph-XYZ-image*, we use a single randomly sampled input view. For *VF* and *PlaNet*, we use a fixed top-down view for both training and testing as we found they only work reasonably well with a fixed viewpoint. Details of the experiment settings are included in Appendix (Section C.3). Our model outperforms all baselines by a large margin. We include videos of pushing object to desired locations in the presence of multiple obstacles in the supplementary file.

Table 2.3: Success rate for pushing objects to target locations.

graph-XYZ [13]	*graph-XYZ-image* [317]	*VF*[59]	*PlaNet*[87]	Ours	Ours-Real
0.76	0.70	0.32	0.16	**0.86**	0.78

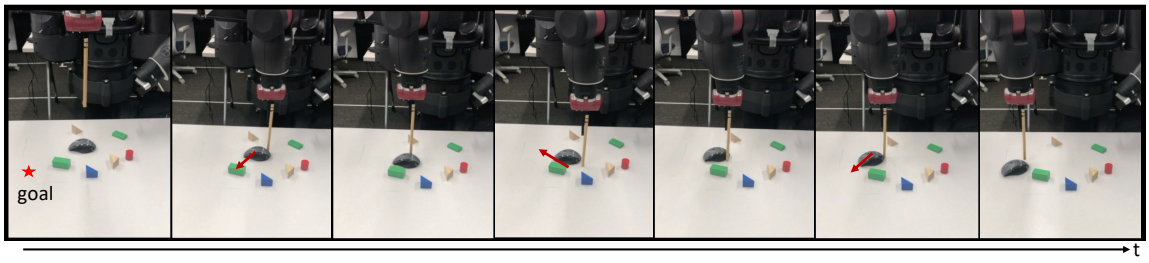


Figure 2.4: **Collision-free pushing on a real-world setup.** The task is to push a mouse to a target location without colliding into any obstacles. Our robot can successfully complete the task with 3 push attempts.

Sim-to-Real Transfer We train our model solely in simulation and test it on object pushing control tasks on a real Baxter platform equipped with a rod-shaped end-effector, similar to the setting in the Bullet simulation. We attached a Intel RealSense D435 RGB-D camera to the robot's left hand, and use only one RGB-D view as input for this experiment. The pose of the camera is different from those seen during training. Please refer to Appendix (Section C.3) for details of our real-world setup, objects selection, and 3D detector training. We report the success rate of real-world pushing in Table 2.3 (Ours-Real). Our model achieves similar success rates for pushing in simulation and in the real world. Since geometry information is shared by simulation and the real world by a large extent, and our model combines the viewpoint-invariant property of the geometry-aware representation and an object-factorized structure, it presents good sim-to-real transferrability.

2.5 Conclusion

We have presented 3D-OES , dynamics models that predict 3D object motion in a 3D latent visual feature space inferred from 2.5D video streams. We empirically showed our model can generalize across camera viewpoints and varying number of objects better than existing 2D dynamics models or dynamics models over 3D object

centroids. To the best of our knowledge, this is the first model that can predict 3D object dynamics directly from RGB-D videos and generalize across scene variations. Our model currently has three main limitations: **(i)** It requires ground-truth 3D object locations and orientations at training time. Automatically inferring those with 3D tracking would permit our model to be trained in a self-supervised manner. **(ii)** It assumes rigid object interactions. Learning dynamics of soft bodies and fluids would require forecasting dense 3D motion fields, or considering sub-object (particle) graphs. **(iii)** It is deterministic. Handling stochasticity via stochastic models or objectives would permit to learn more complex and multimodal object motions.

2.6 Additional Details and Results

2.6.1 Neurally Rendered Physics Simulations from Multiple Views

We show in Figure 2.5 rendered physics simulation videos using the proposed model. The latent 3D feature map of the proposed model is interpretable in the sense that we can render human-interpretable RGB images from the feature map using the learned neural image decoder. More importantly, we can render such simulation videos from any arbitrary view, and the videos captured from different views are consistent with each other.

2.6.2 Counterfactual Experiments

In Figure 2.6, we show more results of conducting counterfactual experiments using the learned neural simulator. We can move objects to arbitrary position, and change their size by moving their features explicitly in the latent 3D feature space. Although the model has never been trained on this task, it can generate reasonable simulation results after such manipulations on the objects.

2.6.3 Data Collection Details

Here we describe details of the data used in Section 4.

Pushing Our training data contains RGB-D video streams where the robot pushes objects which in turn can collide and push other objects on the table. We create scenes using 31 different 3D object meshes, including 11 objects from the MIT Push dataset [320] and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset [28]. We split our dataset so that 24 objects are used during training. At test time, we evaluate the prediction error on the remaining 7 objects. At training time each scene contains at most two (potentially interacting) objects. At test time, we vary the number of objects from one up to five. We randomize the textures of the objects during training to improve transferability to the real world [272]. We consider a simulated Kuka robotic arm equipped with a single rod (as shown in Figure 2.3. The objects can move on a planar table surface of size $0.6m \times 0.6m$ when pushed by the arm, or by other objects. We collect training interaction trajectories by instantiating the gripper nearby a (known) 3D object segmentation mask. We sample random pushing action sequences with length of 5 timesteps, where each action is a horizontal displacement of the robot's end-effector ranging from $3cm$ to $6cm$, and each timestep is defined to be 200ms. We record objects displacement 1 sec after the push. We place cameras at 27 nominal different views including 9 different azimuth angels ranging from the left side of the agent to the right side of the agent combining with 3 different elevation angles from 20, 40, 60 degrees. All cameras are looking at the 0.1m above the center of the table, and are 1 meter away from the look-at point. At each timestep, all cameras are purturbed randomly around their nominal viewpoints, and we record all 27 views. At training time, our model consumes three randomly selected concurrent camera viewpoints as input. At test time, we use the 3D object detector to predict the 3D object segmentation mask, and our model is tested with either three or a single view as input, all randomly selected. All images are 128×128. There are 5000 pushing trajectories in the training data, and 200 pushing trajectories in the test data.

Falling We use the 3D meshes of the block objects introduced in [116], which includes cones, cylinders, rectangles, tetrahedrons, and traingles with a variety of shapes. We

randomly select 1-3 objects and initialize their position by placing them on the table surface, and let one object falls freely from the air. One timestep is defined to be 40ms. All other settings are identical to the settings for pushing.

2.6.4 Additional Experimental Details

Implementation Details of the 3D Object Graph Neural Networks

We use ground-truth 3D bounding boxes for cropping the scene feature map $\mathbf{M}$ at training time, and 3D predicted boxes provided by our 3D detector at test time. The loss function is the summation of the L2 distance between the predicted and GT translation, and the L2 distance between the predicted and GT quaternions. The inputs to the graph networks are the cropped 3D feature maps of each objects with the size of 16×16. We first transforms the object-center 3D feature map into a feature vector with three 3D-conv layers followed by an average pooling layer and two FC layers of size 32 with leaky-relu. The vectorized object features are then concatenated with the position and orientation of the objects as inputs to a standard graph network. In the graph network, both the node and edge encoders are 4-layer MLPs with layer size of 32 and leaky-relu activation. Our model is initialized with Xavier initialization and trained using the Adam optimizer for 90K steps. We train two separate models, one for pushing and one for falling.

For the 2D-to-3D image encoding using GRNNs [276], we follows the exact neural architecture as in [276], which takes as input RGB-D images and outputs 3D feature map $\mathbf{M}_t$ of size $64 \times 64 \times 64 \times 32$. Our detector also follows the architecture design of [276], which extends the 2D faster RCNN architecture to predict 3D bounding boxes from 3D features maps, as opposed to 2D boxes from 2D feature maps. The detector takes the output 3D feature map from the 2D-to-3D lifting as input to predict object bounding boxes. The detector consists of one down-sampling layer and three 3D residual blocks, each having 32 channels. We use 1 anchor box at each grid location in the 3D feature map with a size of 0.12 meters. The detector predicts an objectness score for each anchor box and selects boxes that exceeds a threshold. We set the threshold to be 0.9. We train the object detector with all the frames in the training data.

Implementation Details for Baselines

Here we describe the baselines discussed in Section 4 in detail.

1. *graph-XYZ*, a model that uses the 3D object centroid (X, Y, Z) as object state, and incorporate cross-object interactions for forecasting 3D translation using graph convolutions over a object graph, similar to [8, 296]. Since the canonical pose of an object is undefined, object orientation is not included in the object state. This model neglects object shape and appearance. The graph networks used in all baselines follow the exact design as the one we use in our model (4-layer MLPs for both the node and edge encoder). The only difference is that its inputs do not contain any object appearance features.
2. *graph-XYZ-image*, a model that uses the 3D object centroid (X, Y, Z) and object-centric 2D image feature embeddings for forecasting 3D translation. This baseline model extracts 2D CNN features from each image, concatenates the features with the camera viewpoint, and transforms the combined features into an object appearance feature vector. The feature vector is concatenated with the 3D object centroid and fed into a graph network (identical to the one used in *graph-XYZ*) to predict future object 3D translation. When taking multiple views as inputs, the model takes the average of the appearance feature vectors across views.
3. Visual Foresight (*VF*) [59], a model that uses the current frame and the action of the agent to predict future 2D frames by "moving pixels" based on predicted 2D pixel flow fields. It is based on the publicly available code of [59] that uses such frame predictive model to infer an action trajectory that brings an object pixel to the desired (2D) location in the image space.
4. *PlaNet*[87], a model that learns a scene-level embedding by predicting future frames and the reward given the current frame. *PlaNet* only deals with single-goal tasks and does not apply to our multi-goal pushing task. We extend it to our setting by appending the goal state to the observation. In practice, we augment the latent state vector produced from its state encoder's first fully connected layer with a randomly selected goal, and provide the model with reward computed correspondingly. The reward at each timestep is the computed as the negative of the distance-to-goal.

Note that both *VF* and *PlaNet* are self-supervised models that do not require ground-truth object states during training. However, we believe that since such supervision is readily accessible in simulation, we should leverage them to push the performance of the learned dynamics model. Self-supervised models are more favored when trained directly in the real world, where strong supervisions are not available, but as we showed in our experiments, our model trained solely in simulation can transfer reasonably well to the real world without any fine-tuning. As a result, we believe including the comparison with such self-supervised baselines is arguably fair and reasonable.

Details for Pushing with MPC

Here we described details of pushing with MPC discussed in section 4.3.

Pushing without obstacle We test the performance of our model with MPC to push objects to desired locations. We run 50 experiments in the Bullet simulator. For each testing sample, we place either 1 or 2 objects in the $0.6m \times 0.6m$ workspace randomly, and sample a random goal for each object. The maximum distance of the goal to the initial position for each object is capped at $0.25m$. For our model and *graph-XYZ-image*, we use a single randomly sampled view. For *VF* and *PlaNet*, we use a fixed top-down view for both training and testing. We set the maximum number of steps for each action sequence to be 10, and evaluate 30 random action sequences before taking an action. We use planning horizon of 1 since greedy action selection suffices for this task. The results are reported in Table 2.3. Note that we also train and test variants of VF and PlaNet to take observations from varying camera viewpoints, together with camera pose information. However, they both fail completely on this task.

Collision-free pushing In order to test our model's multi-step prediction performance, we evaluate our model on pushing in scenes with randomly sampled obstacles, and the robot is required to push an object to desired goal without colliding into any obstacle. For quantitative evaluation, we randomly place an object of interest

and a goal position in the planar workspace. One obstacle object is placed between them with a small perturbation, so that there exists no straight collision-free path to reach the goal. The distance from the object to its goal is uniformly sampled from the range $[0.24m, 0.40m]$. Similarly, we run 50 examples, and use only one randomly selected camera view as input to our model. We evaluate 60 randomly sampled action sequences with length of 25 steps, and use a planning horizon of 10 steps. We achieve a success rate of **0.68** for this task.

Since randomly placing *multiple* obstacles in the scene for quantitative evaluation while ensuring existence of collision-free path is non-trivial, we show qualitative planning results for such complex scenes in the supplementary video.

For both with- and without- obstacle pushing, it is considered a successful pushing sequence if all objects end up within 4cm (about half of the average object size) from the target positions on average.

Sim-to-real transfer for pushing in the real world We use a Baxter robot equipped with a rod-shaped end-effector attached to its right hand, similar to the setting in the Bullet simulation. One Intel RealSense D435 RGB-D camera is attached to the robot's left hand, and we use only one view for our experiment, as shown in Figure 2.7. Please refer to the supplementary video for more qualitative results.

Due to reachability considerations, we down-scaled the size of the planar workspace by twice from the one in simulation, resulting a workspace of $0.3m \times 0.3m$. For a fair comparison, we also down-scaled with the same factor the object-to-goal distance, length of horizontal movement per action step, and size of the tolerance for determining success/failure. We pick 20 objects with size of 5 to $10cm$, which are commonly seen in a office setting, including fruits, wooden blocks, and stationery, and evaluate 5 pushing samples for each of them. Some of objects selected are shown in Figure 2.8.

For object detection in the real-world, we train our 3D detector using simulated data, and fine-tune it using a small set of real data (100 images capturing 25 distinct object configurations) collected using 4 cameras. The ground truth bounding-boxes and segmentation masks are obtained via background subtraction.

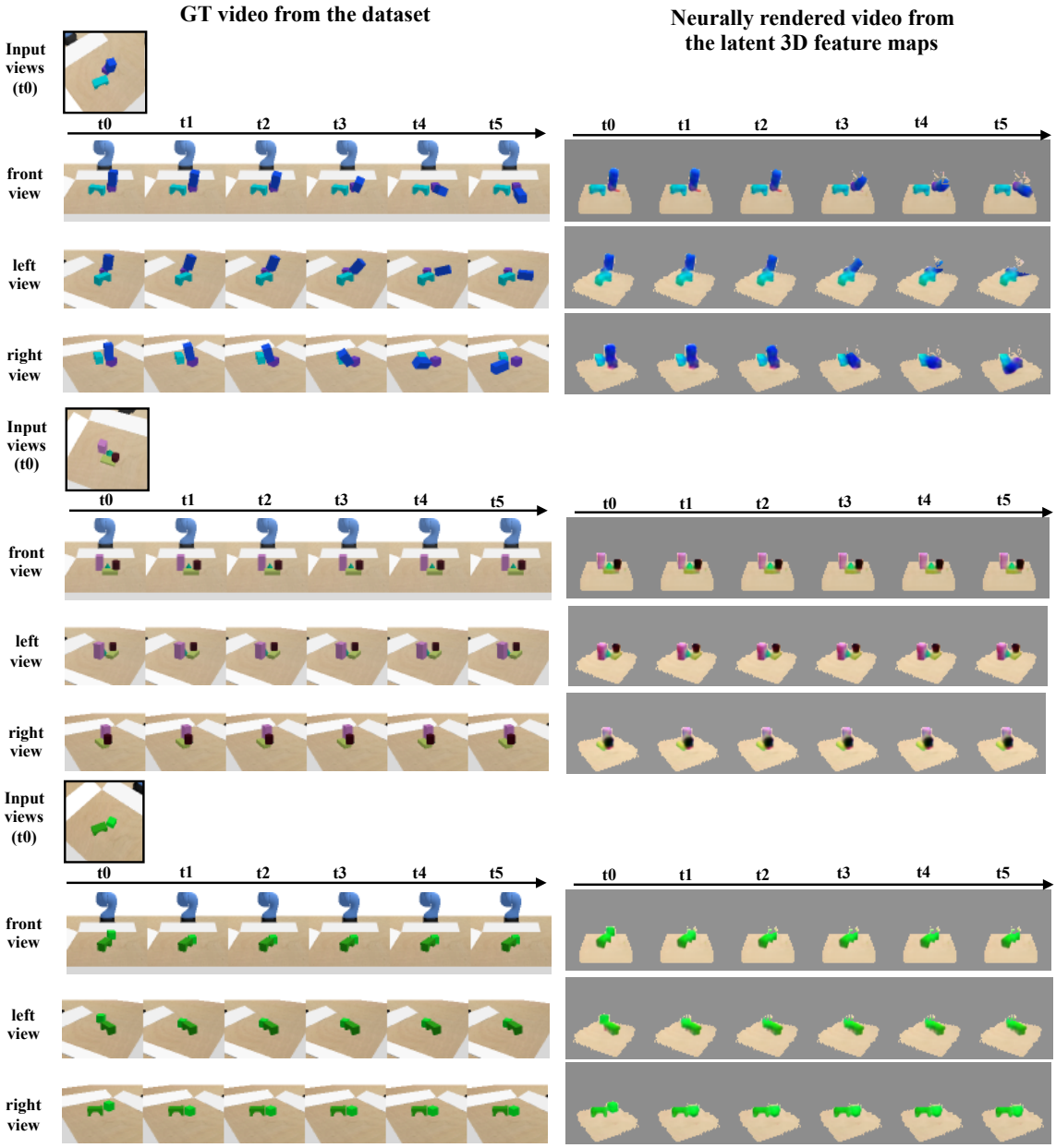


Figure 2.5: **Neurally rendered simulation videos from three different views** Left: groundtruth simulation videos from the dataset. The simulation is generated by the Bullet Physics Simulation. Right: neurally rendered simulation video from the proposed model. Our model forcasts the future latent feature by explicitly warping the latent 3D feature maps, and we pass these warped latent 3D feature maps through the learned 3D-to-2D image decoder to decode them into human interpretable images. We can render the images from any arbitrary views and the images are consistent across views.

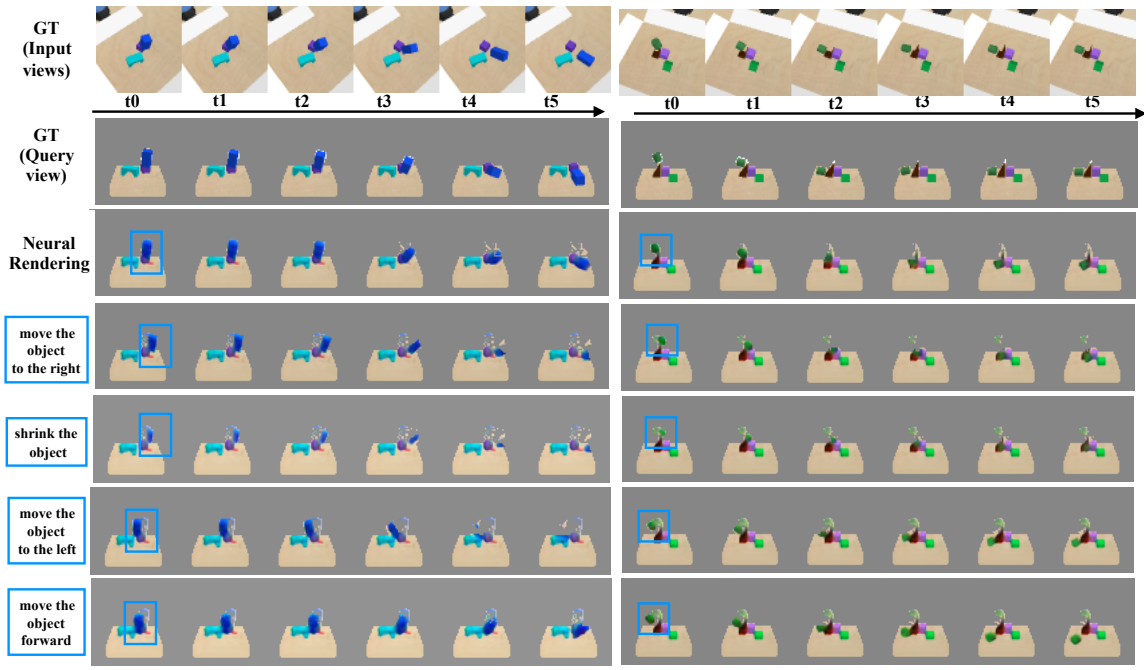


Figure 2.6: **Neurally rendered simulation videos of counterfactual experiments.** The first row shows the ground truth simulation video from the dataset. Only the first frame in this video is used as input to our model to produce the predicted simulations. The second row shows the ground truth simulation from a query view. Note that our model can render images from any arbitrary view. We choose this particular view for better visualization. The third row shows the future prediction from our model given the input image. The following rows show the simulation after manipulating an objects (in the blue box) according the instruction on the left most column.

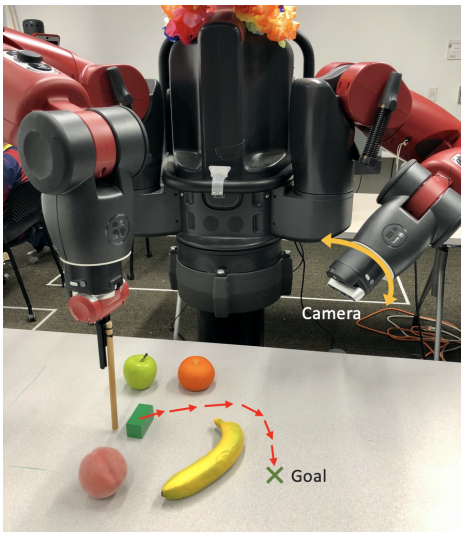


Figure 2.7: Real-world setup with Baxter

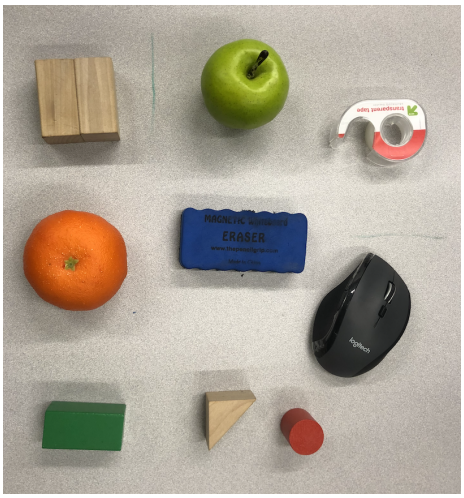


Figure 2.8: Objects for real-world experiments

Chapter 3

Dynamics Model Adaptation with Hypernetworks

3.1 Introduction

Humans learn dynamics models that predict results of their interactions with the environment, and use such predictions for selecting actions to achieve intended goals [92, 183]. These models capture intuitive physics and mechanics of the world and are remarkably versatile: they are expressive and can be applied to all kinds of environments that we encounter in our daily lives, with varying dynamics and diverse visual and physical properties. In addition, humans do not consider these models fixed over the course of interaction; we observe how the environment behaves in response to our actions and quickly adapt our model for the situation at hand based on new observations. Let us consider the scenario of moving an object on the ground. We can infer how heavy the object is by simply looking at it, and we can then decide how hard to push. If it does not move as much as expected, we might realize it is heavier than we thought and increase the force we apply [88].

Motivated by this, we propose *HyperDynamics*, a dynamics meta-learning framework for that generates parameters for dynamics models (*experts*) dedicated to the situation at hand, based on observations of how the environment behaves. HyperDynamics has three main modules: i) an encoding module that encodes a few

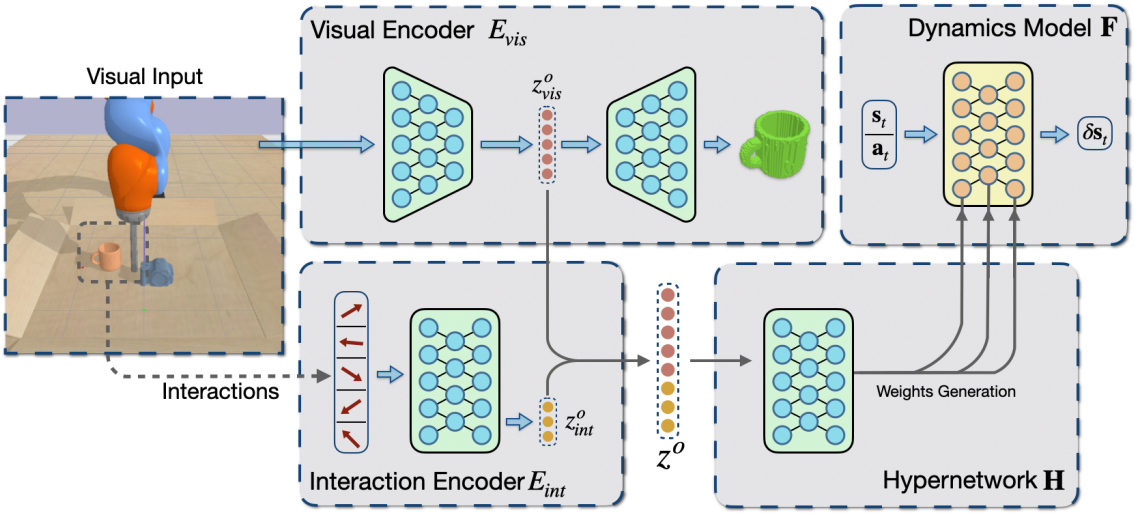


Figure 3.1: HyperDynamics encodes the visual observations and a set of agent-environment interactions and generates the parameters of a dynamics model dedicated to the current environment and timestep using a hypernetwork. HyperDynamics for pushing follows the general formulation, with a visual encoder for detecting the object in 3D and encoding its shape, and an interaction encoder for encoding a small history of interactions of the agent with the environment.

agent-environment interactions and the agent's visual observations into a latent feature code, which captures the properties of the dynamical system, ii) a hypernetwork [83] that conditions on the latent feature code and generates parameters of a dynamics model dedicated to the observed system, and iii) a target dynamics model constructed using the generated parameters that takes as input the current low-dimensional system state and the agent action, and predicts the next system state, as shown in Figure 3.1. We will be referring to this target dynamics model as an *expert*, as it specializes on encoding the dynamics of a particular scene at a certain point in time. HyperDynamics conditions on real-time observations and generates dedicated expert models on the fly. It can be trained in an end-to-end differentiable manner to minimize state prediction error of the generated experts in each task.

Many contemporary dynamics learning approaches assume a fixed system without considering potential change in the underlying dynamics, and train a fixed dynamics model [11, 67, 292, 333]. Such expert models tend to fail when the system behavior changes. A natural solution to address this is to train a separate expert for each

dynamical system. However, this can hardly scale and doesn't transfer to systems with novel properties. Inspired by this need, HyperDynamics aims to infer a system's properties by simply observing how it behaves, and automatically generate an expert model that is dedicated to the observed system. In order to address system variations and obtain generalizable dynamics models, many other prior works propose to condition on visual inputs and encode history of interactions to capture the physical properties of the system [59, 63, 87, 140, 207, 233, 311, 312]. These methods attempt to infer system properties; yet, they optimize a single and fixed global dynamics model, which takes as input both static system properties and fast-changing states, with the hope that such a model can handle variations of systems and generalize. We argue that a representation which encodes system information varies across different system variations, and each instance of this presentation ideally should correspond to a different dynamics model that needs to be used in the corresponding setting. These models are different, but also share a lot of information as similar systems will have similar dynamics functions. HyperDynamics makes explicit assumptions about the relationships between these systems, and attempts to exploit this regularity and learn such commonalities across different system variations. There also exsists a family of approaches that attempt online model adaptation via meta-learning [42, 65, 190, 192]. These methods perform online adaptation on the parameters of the dynamics models through gradient descent. In this work, we empirically show that our approach adapts better than such methods to unseen system variations.

We evaluate HyperDynamics on single- and multi-step state predictions, as well as downstream model-based control tasks. Specifically, we apply it in a series of object pushing and locomotion tasks. Our experiments show that HyperDynamics is able to generate performant dynamics models that match the performance of separately and directly trained experts, while also enabling effective generalization to systems with novel properties in a few-shot manner. We attribute its good performance to the multiplicative way of combining explicit factorization of encoded system features with the low-dimensional state representation of the system. In summary, our contributions are as follows:

- We propose HyperDynamics, a general dynamics learning framework that is able to generate expert dynamics models on the fly, conditioned on system properties.

- We apply our method to the contexts of object pushing and locomotion, and demonstrate that it matches performance of separately trained system-specific experts.
- We show that our method generalizes well to systems with novel properties, outperforming contemporary methods that either optimize a single global model, or attempt online model adaptation via meta-learning.

3.2 Related Work

3.2.1 Model learning and model adaptation

To place HyperDynamics in the context of existing literature on model learning and adaptation, we distinguish four groups of methods, depending on whether they explicitly distinguish between dynamic—such as joint configurations of an agent or poses of objects being manipulated—and static —such as mass, inertia, 3D object shape—properties of the system, and whether they update the dynamics model parameters or the static property values of the system.

(i) No adaptation. These methods concern dynamics model over either low-dimensional states or high-dimensional visual inputs of a specific system, without considering potential changes in the underlying dynamics. These models tend to fail when the system behavior changes. Such *expert* dynamics models are popular formulations in the literature [11, 67, 292, 333]. They can be adapted through gradient descent at any test system, yet that would require a large number of interactions.

(ii) Visual dynamics and recurrent state representations. These methods operate over high-dimensional visual observations of systems with a variety of properties [59, 63, 140, 178, 198, 207, 312], hoping the visual inputs could capture both the state and properties of the system. Some also attempt to encode a history of interactions with a recurrent hidden state [59, 87, 140, 233, 311], in order to implicitly capture information regarding the physical properties of the system. These methods use a single and fixed global dynamics model that takes system properties as input directly, together with its state and action.

(iii) System identification [16]. These methods model explicitly the properties of the dynamical system using a small set of semantically meaningful physics parameters,

such as mass, inertia, damping, etc. [5, 187], and adapt by learning their values from data on-the-fly [12, 32, 145, 232, 233].

(iv) Online model adaptation via meta-learning [42, 65, 190, 192]. These methods perform online adaptation on the parameters of the dynamics models through gradient descent. They are optimized to produce a good parameter initialization that can be adapted to the local context with only a few gradient steps at test time.

The expert models in (i) work well for their corresponding systems but fail to handle system variations. Methods in (ii) and (iii), though attempting to infer system properties, optimize a single and fixed global dynamics model, which takes as input both static system properties and fast-changing states. Methods in (iv) attempt online update for a learned dynamics model. HyperDynamics differs from these methods in that it generates system-conditioned dynamics models on the fly. In our experimental Section, we compare with all these methods and empirically show that our method adapts better to unseen systems.

3.2.2 Background on Hypernetworks

Central to HyperDynamics is the notion of using one network (the hypernetwork) to generate the weights of another network (the target network), conditioned on some forms of embeddings. The weights of the target network are not model parameters; during training, their gradients are backpropagated to the weights of the hypernetwork [30]. This idea is initially proposed in [83], where the authors demonstrate leveraging the structural features of the original network to generate network layers, and achieve model compression while maintaining competitive performance for both image recognition and language modelling tasks. Since then, it has been applied to various domains. In Ratzlaff and Fuxin [221] the authors use hypernetworks to generate an ensemble of networks for the same task to improve robustness to adversarial data. Platanios et al. [213] attempts to generate language encoders conditioned on the language context for better weight sharing across languages in a machine translation setting. Other applications include but are not limited to: weight pruning [167], multi-task learning [131, 182, 243], neural architecture search [19, 330], and continual learning [281]. To the best of our knowledge, this is the first work that applies the idea of generating model parameters to the domain of model-based RL, and proposes

to generate expert models conditioned on system properties.

3.3 HyperDynamics

HyperDynamics generates expert dynamics models on the fly, conditioned on observed system features. We study applying HyperDynamics to a series of object pushing and locomotion tasks, but in principle it is agnostic to the specific context of a dynamics learning task. In this section, we first present the general form of HyperDynamics, and then describe the task-specific modules we used.

Given a dynamical system o (which is typically composed of an agent and its external environment) with a certain set of properties, HyperDynamics observes both how it behaves under a few actions (interactions), as well as its visual appearance when needed. Then, a visual encoder E_{vis} encodes the visual observation I into a latent visual feature code z^o_{vis}, and an interaction encoder E_{int} encodes a k-step interaction trajectory $\tau = (s_0, a_0, ...s_k, a_k)$ into a latent physics feature code z^o_{int}. We then combine these two features codes by concatenating them, $z^o = [z_{int}, z_{vis}]$, forming a single vector that captures the underlying dynamics properties of the system. A *hypernetwork* $\mathbf{H}$ maps z^o to a set of neural network weights for a forward dynamics model $\mathbf{F}^o$, which is a mapping from the state of the system and the action of the agent, to the state change at the subsequent time step $t+1$: $\delta s_t = \mathbf{F}^o(s_t, a_t)$. The hypernetwork is a fully-connected network and generates all parameters in a single-shot as the output of it's final layer. The detailed architecture of HyperDynamics for an object pushing example is shown in Figure 3.1.

We meta-train HyperDynamics by backpropagating error gradients of the generated dynamics model's prediction, all the way to our visual and interaction encoders and the hypernetwork. Let $\mathcal{O}$ denote the pool of all systems used for training. Let T denote the length of the collected trajectories. Let N denote the number of trajectories collected per system. The final prediction loss for our model is defined as:

$$\mathcal{L}_{pred} = \frac{1}{|\mathcal{O}|NT} \sum_{o=1}^{|\mathcal{O}|} \sum_{i=1}^{N} \sum_{t=1}^{T} \|\delta s_t - \mathbf{F}(s_t, a_t; \mathbf{H}([E_{vis}(I;\phi), E_{int}(\tau;\psi)];\omega))\|_2, \quad (3.1)$$

where ϕ, ψ and ω denote the learned parameters of E_{vis}, E_{int} and $\mathbf{H}$, respectively.

We consider a set of locomotion tasks for which only the interaction history is used as input, in accordance with prior work [e.g., 42, 65, 190, 192], and a pushing task on a diverse set of objects, where the visual encoder encodes the 3D shape of the object.

3.3.1 HyperDynamics for Object Pushing

For our object pushing setting, we consider a robotic agent equipped with an RGB-D camera to observe the world and an end-effector to interact with the objects. The agent's goal is to push objects to desired locations on a table surface. At time t, the state of our pushing system is $s_t^o =< s_t^{obj}, s_t^{rob} >$, where the object state s_t^{obj} is a 13-vector containing the object's position $\mathbf{p}_t^{obj}$, orientation $\mathbf{q}_t^{obj}$ (represented as a quaternion), as well as its translational and rotational velocities $\mathbf{v}_t^{obj}$, and the robot state s_t^{rob} is a 3-vector representing the position of its end-effector. For controlling the robot, we use position-control in our experiments, so the action $a_t = (\delta x, \delta y, \delta z)$ is a 3-vector representing the movement of the robot's end-effector.

Objects behave differently under pushing due to varying properties such as shape, mass, and friction relative to the table surface. For this task, HyperDynamics tries to infer object properties from its observations and generate a dynamics model specific to the object being manipulated. While the 3D object shape can be inferred by visual observations, physics properties such as mass and friction are hard to discern visually. Instead, they can be inferred by observing the motion of an object as a result to a few interactions with the robot [312]. To produce the code z^o, we use both E_{vis} for encoding shape information and E_{int} for encoding physics properties.

Given an object to be pushed, we assume its properties stay unchanged during the course of manipulation. To produce the interaction trajectory τ, the robot pushes the object by k actions sampled from random directions. E_{int} then maps τ to a latent code $z_{int} \in \mathbb{R}^2$. In practice we found $k = 5$ sufficient to yield informative z_{int} for pushing, and using a bigger value does not help further improve performance. We use a simple fully-connected network for E_{int}.

Given a single RGB-D image I of the scene, E_{vis} produces object-centric $z_{vis} \in \mathbb{R}^8$ to encode the shape information of the objects in the scene. We build our E_{vis} using the recently proposed Geometry-Aware Recurrent Networks (GRNNs) [276], which learn to map single 2.5D images to complete 3D feature grids $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$

that represent the appearance and the shape of the scene and its objects. GRNNs learn such mappings using differentiable unprojection (2.5D to 3D) and projection (3D to 2.5D) neural modules and can be optimized end-to-end for both 3D object detection and view prediction, and complete missing or occluded shape information while optimized for such objectives. It helps us implicitly handle possible minor occlusions caused by the robot end-effector during data collection. GRNNs detect objects using a 3D object detector that operates over its latent 3D feature map $\mathbf{M}$ of the scene. The object detector maps the scene feature map $\mathbf{M}$ to a variable number of axis-aligned 3D bounding boxes of the objects (and their segmentation masks if needed). Its architecture is similar to Mask R-CNN [93] but it uses RGB-D inputs and produces 3D bounding boxes. We refer readers to Tung et al. [276] for more details.

E_{vis} first detects all the objects in the scene, and then crops the scene map around the object we want to push to produce an object-centric 3D feature map $\mathbf{M}^{obj} = \text{crop}(\mathbf{M})$, which is then used for shape encoding. We opt for such 3D representations since it allows a more realistic pushing setting: we do not require the camera viewpoint to be fixed throughout training and testing, and we can handle robot-object occlusions better since our encoder learns to complete missing (occluded) shape information in the map $\mathbf{M}^{obj}$. We train E_{vis} for object detection jointly with the prediction loss in Equation 3.1. We also add additional regularization losses by reconstructing the object's shape, where the visual code z_{vis} is passed into a decoder D_{vis} to reconstruct the object shape V^{obj}. Note that the reconstruction loss can be applied from pure visual data, by observing objects from multiple viewpoints, without necessarily interacting with them. This is important because moving a camera around objects is cheaper than interacting with them through the robot's end-effectors. In addition, since our tasks concern planar pushing, we found reconstructing a top-down 2D shape instead of the the full 3D one suffices for regularization, essentially predicting a top-down view of the object from any perspective viewpoint.

Object 3D orientation: context or state? The shape of rigid objects does not change over time; rather, its orientation changes. The framework discussed so far intuitively divides the input information among different modules: the hypernetwork generates a dynamics model conditioned on object properties, and the dynamics model takes care of low-dimensional object states. Then, one natural choice is that the

object state includes both its position and orientation information, and z_{vis} encodes the object shape information oriented at its canonical orientation, independent of its actual orientation. However, when there's only a few objects used for training, the hypernetwork will only strive to optimize for a discrete set of dynamics models, as it only sees a discrete set of object-specific z_{vis}, thus potentially leading to severe overfiting on the training data. Instead of producing z_{vis} to encode object shape information at its canonical pose, we propose to encode the object shape oriented at its current orientation at each time step, and discard the orientation information from the states fed into the generated dynamics model, as their generated parameters would already encode the orientation information. This way, we move the orientation information from the state input (fed into the generated dynamics model) to the shape input (encoded into z_{vis} and fed into the hypernetwork **H**). Using oriented as opposed to canonical shape as input permits **H** to have way more training examples, even from a small number of objects with distinct shapes. Intuitively, **H** now treats the same object with different orientations as distinct objects, and generates a different expert for each. This helps smoothen and widen the data distribution seen by **H**.

3.3.2 HyperDynamics for Locomotion

For locomotion tasks, we consider a legged robot walking in a dynamic environment with changing terrains, where their properties are only locally consistent. At timestep t, the state contains the joint velocities and joint positions of the robot, as well as the position of its center of mass. The dimension of the state depends on the actual morphology of the robot. Torque-control is used to send low-level torque commands to the robot.

Unlike the object pushing setting where the properties of the objects can be safely assumed to remain constant, here the system dynamics changes rapidly due to varying terrain properties. Hence, we adopt an online adaptation setting where at timestep t, the interaction data is taken to be the most recent trajectory $\tau = (s_{t-k}, a_{t-k}, ...s_{t-1}, a_{t-1})$. Similarly, we use a simple fully-connected network to encode the interaction trajectory. In practice, we found $k = 16$ (equal to a trajectory segment of around 0.3 seconds) to be informative enough for E_{int} to infer accurate terrain properties, and using a bigger value does not provide further performance gain. The

encoded $z_{int} \in \mathbb{R}^2$ is then used by $\mathbf{H}$ to generate a dynamics model for the local segment of the terrain in real-time.

3.3.3 Model Unrolling and Action Planning

Action-conditioned dynamics models can be unrolled forward in time for long-term planning and control tasks. We apply our approach with model predictive control (MPC) for action planning and control. Specifically, we use random shooting (RS) [191] for action planning, unroll our model forward with randomly sampled action sequences, pick the best performing one, and replan at every timestep using updated state information to avoid compounding errors.

The unrolling mechanism of HyperDynamics for pushing is straightforward: at each timestep t, a generated dynamics model $\mathbf{F}_t^o$ predicts δs_t^o for the system. Afterwards, we compute $s_{t+1}^o = s_t^o \oplus \delta s_t^o$, where $\oplus$ represents simple summation for positions and velocities, and quaternion composition for orientations. We then re-orient the object-centric feature map $\mathbf{M}^{obj}$ using the updated object orientation $\mathbf{q}_{t+1}^{obj}$ and generate a new dynamics model $\mathbf{F}_{t+1}^o$. Finally, the updated full state s_{t+1}^o, excluding $\mathbf{q}_{t+1}^{obj}$, is used by $\mathbf{F}_{t+1}^o$ for motion prediction. For locomotion, at each timestep we obtain one z_{int}, generate a dynamics model $\mathbf{F}_t^o$ for the local context, and use it for all unrolling steps into the future, since future terrain properties are not predictable.

3.4 Experiments

Our experiments aim to answer these questions: (1) Is HyperDynamics able to generate dynamics models across environment variations that perform as well as expert dynamics models, which are trained specifically on each environment variation? (2) Does HyperDynamics generalize to systems with novel properties? (3) How does HyperDynamics compare with methods that either use fixed global dynamics models or adapt their parameters during the course of interaction through meta-optimization? We test our proposed framework in the tasks of both object pushing and locomotion, and describe each of them in details below.

3.4.1 Object Pushing

Many prior works that learn object dynamics consider only quasi-static pushing or poking, where an object always starts to move or stops together with the robot's end-effector [2, 63, 140, 211]. We go beyond simple quasi-static pushing by varying the physics parameters of the object and the scene, and allow an object to slide by itself if pushed with a high speed. A PID controller controls the robot's torque in its joint space under the hood, based on position-control commands, so varying the magnitude of the action could lead to different pushing speed of the end-effector. We test HyperDynamics on its motion prediction accuracy for single- and multi-step object pushing, as well as its performance when used for pushing objects to desired locations with MPC.

Experimental Setup. We train our model with data generated using the PyBullet simulator [44]. Our setup uses a Kuka robotic arm equipped with a single rod-shaped end-effector, as shown in Figure 3.1. Our dataset consists of only 31 different object meshes with distinct shapes, including 11 objects from the MIT Push dataset [320] and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset [28]. We split our dataset so that 24 objects are used for training and 7 are used for testing. The objects can move freely on a planar table surface of size $0.6m \times 0.6m$. For data collection, we randomly sample one object with randomized mass and friction coefficient, and place it on the table with a random starting position. The mass is uniformly sampled from $[300, 1000]$, the friction coefficient of the object material is uniformly sampled from $[8e^{-4}, 12e^{-4}]$, and the friction coefficient of the table surface is set to be 10, all using the default units in PyBullet. Then, we instantiate the end-effector nearby the object and collect pushing sequences with length of 5 timesteps, where each action is a random horizontal displacement of the robot's end-effector ranging from $3cm$ to $6cm$, and each timestep is defined to be $800ms$. Our method relies on a single 2.5D image as input for object detection and motion prediction. Note that although collecting actual interactions with objects is expensive and time-consuming, data consisting of only varying shapes are cheap and easily accessible. To ensure z_{vis} produced by the visual encoder E_{vis} captures sufficient shape information for pushing, we train it on the whole ShapeNet [28] for shape autoencoding as an auxiliary task, jointly with the dynamics prediction

task. See Appendix 3.6.1 for more details.

Baselines. We compare our model against these baselines: **(1)** *XYZ*: a model that only take as input the action and states of the object and the agent, without having access to its visual feature or physics properties, similar to Andrychowicz et al. [8] and Wu et al. [297]. **(2)** *Direct*: this baseline uses the same visual encoder and interaction encoder as our model does, and also uses the same input, while it passes the encoded latent code z directly into a dynamics model in addition to the system state. Compared to ours, the only difference is that it directly optimizes a single global dynamics model instead of generating experts. This approach resembles the most common way of conditioning on object features used in the current literature [2, 63, 67, 87, 140, 311, 333], where a global dynamics model takes as input both the state and properties of the dynamical system. **(3)** Visual Foresight (*VF*) [59] and *DensePhysNet* [311]: these two models rely on 2D visual inputs of the current frame, and predict 2D optical flows that transform the current frame to the future frame. These models also use a global dynamics model for varying system properties, and feed system features directly into the model, following the same philosophy of *Direct*. These two baselines are implemented using the architectures described in their corresponding papers. **(4)** *MB-MAML* [192]: a meta-trained model which applies model-agnostic meta-learning to model-based RL, and trains a dynamics model that can be rapidly adapted to the local context when given newly observed data. This baseline also uses the same input as our model does; the only difference is that it uses the interaction data for online model update. **(5)** *Expert-Ens*: we train a separate expert model for each distinct object, forming an ensemble of experts. This baseline assumes access to the *ground-truth* orientation, mass, and friction, and serves as an oracle when tested on seen objects. For unseen objects, we select a corresponding expert from the ensemble using shape-based nearest-neighbour retrieval, essentially performing system identification.

Implementation Details. In order to ensure a fair comparison, all the dynamics models used in *Direct*, *MB-MAML* and *Expert-Ens* use the exactly same architecture as the one generated by HyperDynamics. The interaction encoder is a fully-connected network with one hidden layer of size 8. The hypernetwork is a fully-connected network with one hidden layer of size 16. The visual encoder follows the architecture of [276] and its encoder uses 3 convolutional layers, each followed by a max pooling

Model	Seen Objects		Novel Objects	
	$t = 1$	$t = 5$	$t = 1$	$t = 5$
Expert-Ens	**0.82±0.37**	**4.22±2.21**	1.68±0.79	5.83±2.49
XYZ	1.45±0.62	5.89±2.46	1.72±0.66	6.46±2.92
Direct	1.01±0.41	5.14±2.30	1.24±0.54	5.60±2.59
MB-MAML	1.68±0.61	8.91±3.79	1.76±0.73	9.05±5.98
HyperDynamics	**0.83±0.35**	**4.27±2.24**	**0.93±0.53**	**4.77±2.57**

Table 3.1: Motion prediction error (in centimeters).

Model	Seen Objects		Novel Objects	
	w/o obs	w/ obs	w/o obs	w/ obs
Expert-Ens	**0.92**	**0.72**	0.80	0.44
XYZ	0.80	0.42	0.74	0.44
Direct	0.88	0.62	0.84	0.58
MB-MAML	0.70	0.42	0.68	0.38
VF	0.62	—	0.52	—
DensePhysNet	0.70	—	0.58	—
HyperDynamics	**0.92**	**0.70**	**0.92**	**0.68**

Table 3.2: Pushing success rate.

layer, with a fully-connected layer at the end. It uses kernels of size 5, 3 and 3, with 2, 4, and 8 channels respectively. Its decoder uses 3 transposed convolutional layers, with 16, 8, and 1 channels and the same kernel size of 4. The generated dynamics model is a fully-connected network with 3 hidden layers, each of size 32; the same applies to the dynamics models used in *XYZ*, *Direct* and *Expert-Ens*. We use leaky ReLU as activations for all the modules. We use batch size of 8 and a learning rate of $1e-3$. We collected $50,000$ pushing trajectories for training, and $1,000$ trajectories for testing. All models are trained till convergence for 500K steps.

We evaluate all the methods on prediction error for both single-step ($t = 1$) and multi-step unrolling ($t = 5$) (Table 3.1). We also test their performance for downstream manipulation tasks, where the robot needs to push the objects to desired target locations with MPC, in scenes both with and without obstacles. When obstacles are present, the agent needs to plan a collision-free path. We report their success rates

in 50 trials in Table 3.2. Red numbers in bold denote the best performance across all models, while black numbers in bold represent the oracle performance (provided by *Expert-Ens*). *VF* and *DensePhysNet* are not tested for motion prediction since they work in 2D image space and do not predict explicit object motions, and are not applicable for collision checking and collision-free pushing for the same reason.

Our model outperforms all baselines by a margin on both prediction and control tasks. When tested on objects seen during training, while our model needs to infer orientation information and physics properties via encoding visual input and interactions, it performs on par with the *Expert-Ens* oracle, which assumes access to ground-truth orientation, mass and friction coefficient. This shows that the dynamics model generated by HyperDynamics on the fly is as performant as the separately and directly trained dynamics experts. The *XYZ* baseline depicts the best performance possible when shape and physics properties of the objects are unknown. When tested with novel objects unseen during training, our model shows a performance similar to seen objects, outperforming *XYZ* by a great margin, demonstrating that our model transfers acquired knowledge to novel objects well. *Expert-Ens* on novel objects performs similarly to *XYZ*, suggesting that with only a handful of distinct objects in the training set, applying these dedicated models to novel objects, though after nearest-neighbour searches, yields poor generalization. Performance gain of our model over *Direct* suggests our hierarchical way of conditioning on system features and generating experts outperforms the common framework in the literature, which uses a global model with fixed parameters for varying system dynamics. Our model also shows a clear performance gain over *MB-MAML*, when only 5 data samples of interaction are available for online adaptation. This indicates that our framework, which directly generates dynamics model given the system properties, is more sample-efficient than the meta-trained dynamics model prior that needs to adapt to the context via extra tuning.

3.4.2 Locomotion

Experimental Setup. We set up our environment using the MuJoCo simulator [273]. In particular, we consider two types of robots: a planar half-cheetah and a quadrupedal "ant" robot. For each robot, we consider two environments where the

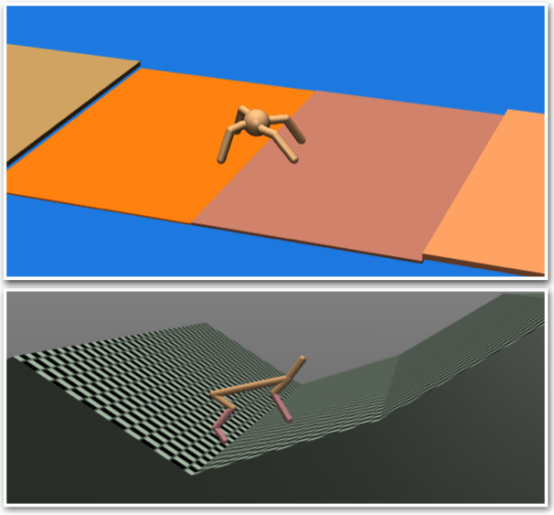


Figure 3.2: We consider two types of robots and two environments for locomotion. Top: Ant-Pier. Down: Cheetah-Slope.

terrains are changing: (1) *Slope*: the robot needs to walk over a terrain consisting of multiple slopes with changing steepness. (2) *Pier*: the robot needs to walk over a series of blocks floating on the water surface, where damping and friction properties vary between the blocks. As a result, we consider these 4 tasks in total: *Cheetah-Slope*, *Ant-Slope*, *Cheetah-Pier* and *Ant-Pier*. Figure 3.2 shows two of them. Unlike the setting of object pushing, randomly and uniformly sampled states in the robot's state space do not match the actual state distribution encountered when the robot moves on the terrain. As a result, on-policy data collection is performed: the latest model is used for action selection to collect best-performing trajectories, which are then used to update the dataset for model training. For *Slope*, the environment is comprised of a series of consecutive terrain segments, each of which has a randomly sampled steepness. During test time, we also evaluate on unseen terrains with either higher or lower steepness. For *Pier*, each of the blocks floating on the water has randomly sampled friction and damping properties, and moves vertically when the robot runs over it, resulting in a rapidly changing terrain. Similarly, during test time we also evaluate using blocks with properties outside of the training distribution. See Appendix 3.6.2 for more details.

Baselines. We compare our model against these baselines: **(1)** *Recurrent* [233]:

Table 3.3: Comparison of average total return for locomotion tasks.

Model	Cheetah-Slope		Cheetah-Pier		Ant-Slope		Ant-Pier	
	Seen	Novel	Seen	Novel	Seen	Novel	Seen	Novel
Expert-Ens	**104.1±59.2**	76.9±42.4	**354.6±182.9**	321.4±171.3	**142.1±77.9**	124.2±71.9	**219.0±113.2**	179.3±87.2
Recurrent	64.4±21.0	76.6±38.1	292.5±132.1	279.7±132.4	**133.9±68.4**	**142.1±66.1**	176.4±87.7	170.5±89.1
MB-MAML	55.3±24.5	62.7±37.7	291.3±143.0	297.1±151.9	104.5±45.2	110.4 ±53.2	180.2±91.2	171.0±87.1
HyperDynamics	**107.9±63.1**	**124.5±64.7**	**349.4±189.0**	**337.2±179.4**	**138.8±65.3**	**140.1±72.0**	**216.6±121.2**	**208.4±103.8**

A recurrent model that also uses GRUs [38] to encode a history of interactions to infer system dynamics and perform system identification implicitly. **(2)** *MB-MAML* [192]: a model meta-trained using MAML, same as the one used for object pushing. It uses the same 16-step interaction trajectory for online adaptation. **(3)** *Expert-Ens*: An ensemble of separately trained experts, similar to the one used for object pushing. The only difference is that in pushing, we train one expert for each distinct object, while now we have a continuous distribution of terrain properties used for training. Therefore, for each task we pick several environments with properties evenly sampled from the continuous range, and train an expert for each of them.

Implementation Details. Again, in order to have a fair comparison, all the forward dynamics models used in *Recurrent* and *MB-MAML* use the same architecture as the one generated by HyperDynamics. Both *Recurrent* and *MB-MAML* use the same input as our model does. Our encoder is a fully-connected network with 2 hidden layers, both of size 128, followed by ReLU activations and a final layer which outputs the encoded z_{int}. The hypernetwork is a fully-connected network with one hidden layer of size 16. The generated dynamics model is a fully-connected network with two hidden layers, each of size 128; the same applies to the dynamics models used in *Recurrent* and *MB-MAML*. During planning and control, 500 random action sequences are sampled with a planning horizon of 20 steps. Data collection is performed in an on-policy manner. We start the first iteration of data collection with randomly sampled actions and collect 10 rollouts, each with 500 steps. We iterate over the data for 100 epochs before proceeding to next iteration of data collection. At each iteration, the size of training data is capped at 50000 steps, randomly sampled from all available trajectories collected. Here we use batch size of 128 and a learning rate of $1e-3$. All models are trained for 150 iterations till convergence. In addition, early stopping is applied if an rolling average of total return decreases.

We apply all the methods with MPC for action selection and control, and report in Table 3.3 the average return computed over 500 episodes. Again, red numbers

denote the best performance and the black ones represent the oracle performance. In all tasks, HyperDynamics is able to infer accurate system properties and generate corresponding dynamics models that match the oracle *Expert-Ens* on seen terrains, and shows a great advantage over it when tested on unseen terrains. *Recurrent* also performs reasonably well on *Ant-Slope*, but our model outperforms both *MB-MAML* and *Recurrent* on most of the tasks. Note that here *Recurrent* can be viewed as serving the same role of the *Direct* baseline in pushing, since it feeds system features together with the states to a fixed global dynamics model. The results suggest that our explicit factorization of system features and the multiplicative way of aggregating it with low-dimensional states provide a clear advantage over methods that either train a fixed global model or perform online parameter adaptation on a meta-trained dynamics prior.

3.5 Conclusion

We presented HyperDynamics, a dynamics meta-learning framework that conditions on system features inferred from observations and interactions with the environment to generate parameters for dynamics models dedicated to the observed system on the fly. We evaluate our framework in the context of object pushing and locomotion. Our experimental evaluations show that dynamics models generated by HyperDynamics perform on par with an ensemble of directly trained experts in the training environments, while other baselines fail to do so. In addition, our framework is able to transfer knowledge acquired from seen systems to novel systems with unseen properties, even when only 24 distinct objects are used for training in the object pushing task. Our method presents explicit factorizations of system properties and state representations, and provides a multiplicative way for them to interact, resulting in a performance gain over both methods that employ a global yet fixed dynamics models, and methods using gradient-based meta-optimization. The proposed framework is agnostic to the specific form of dynamics learning tasks, and we hope this work could stimulate future research into weight generation for adaptation of dynamics models in more versatile tasks and environments. Handling more varied visual tasks, predicting both the architecture and the parameters of the target dynamics model, and applying our method in real-world scenarios are interesting avenues for future work. For pushing,

Figure 3.3: **Top**: 24 objects used for training. **Bottom**: 7 objects used for testing. Objects selected from the ShapeNet Dataset [28] are realistic objects seen during daily life, and objects from the MIT Push dataset [320] consist of basic geometries such as rectangle, triangle, and ellipse. Note that the shape of the testing objects differ from those used for training. (The third *mug* object used for testing is handleless.)

we believe our method has the potential to transfer to real-world without heavy fine-tuning, since it uses a geometry-aware representation, as suggested in Tung et al. [278]. For locomotion, our model should also be easily trainable with data collected in real-world, following the pipeline described in Nagabandi et al. [192].

3.6 Experimental Details

3.6.1 Object Pushing

Our dataset consists of 31 object meshes with distinct shapes, including 11 objects from the MIT Push dataset [320] and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset [28]. 24 objects are used for generating the training data and 7 objects are used for testing (see Figure 3.3). The size of the objects ranges from $7cm$ to $12cm$. For each pushing trajectory, we randomly sample an object with randomized mass and friction coefficient, and place it on a $0.6m \times 0.6m$ table surface at a uniformly sampled random position. At the begining of each trajectory, we instantiate the robot's end-effector nearby the object, with a

random distance from the object's center ranging from $6cm$ to $10cm$. When sampling pushing actions, with a probability of 0.2 the agent pushes towards a completely random direction, and with a probability of 0.8 it pushes towards a point randomly sampled in the object's bounding box. Each action is a displacement with a magnitude sampled from $[3cm, 6cm]$.

Our method is viewpoint-invariant and relies on a single 2.5D image as input. We place cameras at 9 different views uniformly with different azimuth angles ranging from the left side to the right side of the robot, and the same elevation angles of 60 degrees. All cameras are looking at $0.1m$ above the center of the table, and $1m$ away from the look-at point. At test time, our model is tested with a single and randomly selected view. All images are 128×128.

For object pushing with MPC, we place a randomly selected object in the scene with a random initial position, and sample a random target location for it. At each step, we evaluate 30 random action sequences with the model before taking an action. For pushing without obstacle, the maximum distance of the target to the initial position for each object is capped at $0.25m$, and a planning horizon of 1 step is used since greedy action selection suffices for it. For pushing with obstacles, we randomly place 2 additional obstacles in the scene, selected from the same pool of available objects. We place the first obstacle between the starting and goal positions with a small perturbation sampled from $[-2cm, 2cm]$, so that there exists no straight collision-free path from the starting position to the goal. The second obstacle is placed randomly with a distance from the first one ranging in $[15cm, 25cm]$. The distance from the target to the initial position is uniformly sampled from the range $[0.24m, 0.40m]$. The model needs to plan a collision free path and thus a planning horizon of 10 steps is used. For both tasks, we run 50 trials and evaluate the success rate, where it is considered successful if the object ends up within $4cm$ from the target position, without colliding with any obstacles along the way.

Additional model details. HyperDynamics for pushing follows the general formulation discussed in Section 3.3. It's composed of a visual encoder E_{vis}, an interaction encoder E_{int} and a hypernetwork $\mathbf{H}$. These components are trained together to minimize the dynamics prediction loss $\mathcal{L}_{pred}$, as defined in Equation 3.1. Meanwhile, E_{vis} is also trained jointly for object detection and shape reconstruction. Our E_{vis} follows the architecture of GRNNs [276], which uses a similar detection

architecture as Mask R-CNN [93], except that it takes 2.5D input and produces 3D bounding boxes. The detection loss $\mathcal{L}_{det}$ is identical as the one defined in He et al. [93], supervised using ground-truth object positions provided by the simulator. E_{vis} then uses the bounding boxes to crop out a fixed-size feature map for the object under pushing, and encodes it into a visual code z_{vis}. In order to help E_{vis} produce an informative z_{vis}, we added an auxiliary shape reconstruction loss, where z_{vis} is passed into a decoder D_{vis} to reconstruct the object shape V^{obj}. In practice, since our tasks concern planar pushing, we found reconstructing a top-down 2D shape instead of the full 3D one suffices to produce informative z_{vis}. This is essentially performing view prediction for the top-down viewpoint. The network is trained to regress to the ground-truth top-down view of the object, which is provided by the simualtor and does not have any occlusion. The view prediction loss $\mathcal{L}_{vp}$ of GRNNs uses a standard cross-entropy pixel matching loss, where the pixel intensity is normalized. We refer readers to [276] for more details. The final loss for HyperDynamics for pushing reads:

$$\mathcal{L} = \mathcal{L}_{pred} + \mathcal{L}_{det} + \mathcal{L}_{vp} \tag{3.2}$$

Note that while it's expensive to collect actual pushing data with objects, data consisting of various shapes are easily accessible. In order to ensure z_{vis} produced by the shape encoder E_{shape} captures sufficient shape information, we train it on the whole ShapeNet [28] for shape autoencoding (essentially top-down view prediction) to jointly optimize for $\mathcal{L}_{vp}$. In practice, we load random objects in the simulation and collect rendered RGB-D images from randomized viewpoints. During training, data fed into E_{vis} alternates between such rendered data and images of actual objects being pushed, while the other components are only trained with the objects being pushed. Note that such auxiliary data is only used to optimize for $\mathcal{L}_{vp}$. (It's also possible to use such data to optimize for $\mathcal{L}_{det}$, but in our experiments training object detection with only the objects being pushed suffices.) The reason for doing this is that we can collect such visual data very fast using easily accessible object meshes, while we only need to interact with a few objects with actual pushing. This helps our model generalize to novel objects unseen during actual pushing, since E_{vis} is now able to produce a meaningful z_{vis} for these novel objects.

3.6.2 Locomotion

The *Slope* environment consists of a series of consecutive upward slopes with length of $15m$. For training, the height of each slope is randomly sampled from $[0.5m, 3.5m]$, and for testing, the height is randomly sampled from $[0m, 0.25m] \cup [3.75m, 4m]$. The *Pier* environment consists of a series of consecutive blocks with length of $4m$. For training, the damping coefficient of each block is randomly sampled from $[0.2, 0.8]$, and for testing, the damping coefficient is randomly sampled from $[0, 0.1] \cup [0.9, 1.0]$. 5 experts are trained to form the expert ensemble for each task. The *half-cheetah* robot has 6 controllable joints while the *ant* has 10. The reward function for *half-cheetah* is defined to be $\frac{x_{t+1}-x_t}{0.01} - 0.05\|a_t\|_2^2$, and for *ant* it is $\frac{x_{t+1}-x_t}{0.02} + 0.05$, where x_t refers to the x-coordinate of the agent at time t.

3.7 Additional Experimental Results

3.7.1 Model Ablations

In order to evaluate the effect of each network component in our model and motivate our design choices, we present an ablation study here, and report the performance of each ablated model in Table 3.4 and 3.5. For pushing, our model contains a visual decoder for top-down shape reconstruction (top-down view prediction), uses a cropping step to produce object-centric feature maps, and uses $z_{vis} \in \mathbb{R}^8$ and $z_{int} \in \mathbb{R}^2$. For locomotion, our model uses $z_{int} \in \mathbb{R}^2$. We vary the sizes of these latent codes and evaluate how they affect the model performance. In addition, in order to help understand the performance of the 3D detector in our visual encoder, we also report the prediction error using ground-truth object positions as input. The results in the table indicates that for pushing, it's essential to include the visual decoder to regularize the training, otherwise the model overfits badly on the training data. The cropping step is also important and the object-centric feature map helps the model to make more accurate predictions. As for the object detector, when compared to the model using ground-truth object positions, our model using detected object positions only shows a minor performance drop, suggesting that the 3D detector in the visual encoder is able to accurately detect the object locations. The table also motivates our

Table 3.4: Prediction error of HyperDynamics for pushing with different model ablations.

Model	Ours	w/ gt object positions	w/o visual decoder	w/o cropping	$z_{vis} \in \mathbb{R}^{16}$	$z_{vis} \in \mathbb{R}^{32}$	$z_{int} \in \mathbb{R}^{4}$	$z_{int} \in \mathbb{R}^{8}$
Error (seen)	**0.83±0.35**	**0.76±0.32**	0.87±0.44	1.45±0.64	**0.82±0.43**	**0.84±0.37**	**0.83±0.39**	**0.81±0.34**
Error (novel)	**0.93±0.53**	**0.86±0.44**	4.32±2.08	1.76±0.90	**0.94±0.51**	**0.96±0.49**	**0.93±0.52**	**0.94±0.55**

Table 3.5: Average total return of HyperDynamics for locomotion with different model ablations.

Model	Cheetah-Slope		Cheetah-Pier		Ant-Slope		Ant-Pier	
	Seen	Novel	Seen	Novel	Seen	Novel	Seen	Novel
Ours	107.9±63.1	124.5±64.7	349.4±189.0	337.2±179.4	138.8±65.3	140.1±72.0	216.6±121.2	208.4±103.8
$z_{int} \in \mathbb{R}^{4}$	103.5±59.0	127.6±62.9	351.2±176.4	333.5±167.9	141.2±69.3	137.1±73.2	221.2±115.9	202.1±93.0
$z_{int} \in \mathbb{R}^{8}$	112.1±64.0	122.9±59.9	363.7±173.6	329.8±173.1	134.5±63.8	134.9±75.8	214.2±124.1	206.7±97.3

Table 3.6: Comparison of prediction error ($\times 10^{-2}$) for locomotion tasks.

Model	Cheetah-Slope		Cheetah-Pier		Ant-Slope		Ant-Pier	
	Seen	Novel	Seen	Novel	Seen	Novel	Seen	Novel
Expert-Ens	**3.4±0.6**	26.1±6.4	**5.4±0.8**	12.0±6.1	**8.9±3.6**	25.4±9.2	**14.2±4.7**	41.7±13.2
Recurrent	5.2±1.7	22.4±5.2	5.2±1.2	12.1±4.5	**9.1±3.2**	**23.0±8.9**	18.3±8.0	39.1±18.0
MB-MAML	5.7±2.2	21.0±9.2	5.1±1.3	12.4±6.7	12.8±5.2	29.1 ±11.2	18.9±7.2	42.3 ±19.7
HyperDynamics	**3.5±0.6**	**17.9±7.3**	**5.6±1.0**	**11.3±4.7**	**9.3±3.4**	**22.8±7.9**	**14.7±5.3**	**33.2±14.2**

choice for the dimension of the latent code z_{vis} (pushing) and z_{int} (both pushing and locomotion), as further increasing their sizes does not result in any clear performance gain. (In Table 3.5, none of the numbers is marked in red as the results are similar across all ablated models.)

3.7.2 Additional Results on Prediction Error for Locomotion

In the locomotion tasks, the data collected is not i.i.d., since it depends on the model that's being optimized online. In order to show a clear comparison between our method and baselines regarding their prediction accuracy, we train HyperDynamics till convergence for each task, and use this converged model with MPC to collected a fixed dataset. Then, we compare the prediction error of all models on this single dataset and report the results in Table 3.6. The error is computed as the mean squared error between the predicted state and the actual state in the robot's state space. Again, our model is able to match the performance of *Expert-Ens* on the seen terrains and still performs reasonably well on unseen terrains, outperforming the baselines consistently.

Chapter 4

Constructing Differentiable Simulators for Learning to Manipulate Fluids

4.1 Introduction

Imagine you are fishing on the lakeside. Your hat falls into the water and starts to float out of reach. In order to get it back, you use your hands to paddle the water gently, generating a current that pulls the the hat back into reach. In this scenario, the water functions as a transmitting medium for momentum exchange to accomplish the manipulation task. In fact, from creating latte art to making ice creams, humans frequently interact with different types of fluids everyday (Figure 4.1). However, performing these tasks are still beyond the reach of today's robotic systems due to three major challenges. First, fluid systems exhibit a wide range of material models (e.g., air, liquid, granular flow, etc. [18]) that are difficult to simulate within a unified framework. Second, describing the state of a fluid system under manipulation requires a high dimensional state space [155]. Third, the nonlinear dynamics of different coupling models pose unique challenges to efficient solution search in the state space [237]. For example, although water and smoke face the same computational challenges in solving fluid equations, manipulating these two mediums

Figure 4.1: We encounter various manipulation tasks involving fluids in everyday life. Left: real-world scenarios including making latte art, making ice creams and designing air ventilation systems. Right: simulation tasks in FluidLab involving latte art, ice-creams and indoor air circulation.

in different physical contexts requires different strategies.

Previous research in robotic manipulation covering fluids mostly adopts relatively simple task settings, and usually considers tasks with a single-phase fluid, such as pouring water [155, 237, 242] or scooping objects from water [10, 240]. In this work, we aim to study the problem of *complex fluid manipulation* with more challenging task settings and complex goal configurations.

Since virtual environments have played an instrumental role in benchmarking and developing robot learning algorithms [20, 132, 306], it is desirable to have a virtual platform providing a versatile set of fluid manipulation tasks. In this work, we propose *FluidLab*, a new simulation environment to pave the way for robotic learning of complex fluid manipulation skills that could be useful for real-world applications. FluidLab provides a new set of manipulation tasks that are inspired by scenarios encountered in our daily life, and consider both interactions within multiple fluids (e.g., coffee and frothed milk), as well as the coupling between fluids and solids. Building such an environment poses a major challenge to the simulation capability: fluids encountered in our life consist of a wide spectrum of materials with different properties; in addition, simulating these tasks also requires modeling interactions with other non-fluid materials. For example, coffee is a type of (mostly inviscid) *Newtonian* liquid; frothed milk behaves as *viscous Newtonian* liquid (i.e., with a constant viscosity that does not change with stress) (Figure 4.1, R-A), while ice cream is a type of *non-Newtonian* fluid (in particular, a shear-thinning fluid with its viscosity decreasing as the local stress increasing) (Figure 4.1, R-B); when

stirring a cup of coffee, we use a *rigid* stick to manipulate the coffee (Figure 4.1, R-A). However, there still lacks a simulation engine in the research community that supports such a wide variety of materials and couplings between them. Therefore, we developed our own physics engine, named *FluidEngine*, to support simulating the tasks proposed in FluidLab. FluidEngine is a general-purpose physics simulator that supports modeling solid, liquid and gas, covering materials including elastic, plastic and rigid objects, Newtonian and non-Newtonian liquids, as well as smoke and air. FluidEngine is developed using Taichi [102], a domain-specific language embedded in Python, and supports massive parallelism on GPUs. In addition, it is implemented in a fully *differentiable* manner, providing useful gradient information for downstream optimization tasks. Closest to FluidEngine is Nvidia FleX [175], a physics engine that supports both rigid and deformable objects and is widely used in a variety of manipulation environments [72, 155, 237]. However, FleX doesn't support simulating plastic materials or air, and is not differentiable. FluidEngine follows the OpenAI Gym API [20] and also comes with a user-friendly Python interface for building custom environments. We also provide a GPU-accelerated renderer developed with OpenGL, supporting realistic image rendering in real-time.

We evaluate state-of-the-art Reinforcement Learning algorithms and trajectory optimization methods in FluidLab, and highlight major challenges with existing methods for solving the proposed tasks. In addition, since differentiable physics has proved useful in many rigid-body and soft-body robotic tasks [143, 156, 308, 309], it is desirable to extend it to fluid manipulation tasks. However, supporting such a wide spectrum of materials present in FluidLab in a differentiable way is extremely challenging, and optimization using the gradients is also difficult due to the highly non-smooth optimization landscapes of the proposed tasks. We propose several optimization schemes that are generally applicable to the context of fluids, making our simulation fully differentiable and providing useful directions to utilize gradients for downstream optimizations tasks. We demonstrate that our proposed techniques, coupled with differentiable physics, could be effective in solving a set of challenging tasks, even in those with highly turbulent fluid (e.g., inviscid air flow). We summarize our main contributions as follows:

- We propose *FluidLab*, a set of standardized manipulation tasks to help comprehensively study *complex fluid manipulation* with challenging task settings and

potential real-world applications.

- We present *FluidEngine*, the underlying physics engine of FluidLab that models various material types required in FluidLab's tasks. To the best of our knowledge, it is the first-of-its-kind physics engine that supports such a wide spectrum of materials and couplings between them, while also being *fully differentiable*.
- We benchmark existing RL and optimization algorithms, highlighting challenges for existing methods, and propose optimization schemes that make differentiable physics useful in solving a set of challenging tasks. We hope FluidLab could benefit future research in developing better methods for solving complex fluid manipulation tasks.

4.2 Related Work

Robotic manipulation of rigid bodies, soft bodies and fluids Robotic manipulation has been extensively studies over the past decades, while most of previous works concern tasks settings with mainly rigid objects [33, 264, 278, 326]. In recent years, there has been a rising interest in manipulation problems concerning systems with deformable objects, including elastic objects such as cloth [150, 157, 300], as well as plastic objects like plasticine [108]. Some of them also study manipulation problems involving fluids, but mainly consider relatively simple problems settings such as robotic pouring [155, 195] or scooping [10], and mostly deal with only water [10, 145, 172, 237, 240]. In this work, we aim to study a comprehensive set of manipulation tasks involving different fluid types with various properties. It is also worth mentioning that aside from the robotics community, a vast literature has been devoted to fluid control in computer graphics [101, 314, 337], with their focus on artistic visual effects yet neglecting the physical feasibility (e.g., arbitrary control forces can be added to deform a fluid body into a target shape). In contrast to them, we study fluid manipulation from a realistic *robotics* perspective, where fluids are manipulated with an embodied robotic agent.

Simulation environments for robotic policy learning In recent years, simulation environments and standardized benchmarks have significantly facilitated research in robotics and policy learning. However, most of them [20, 132, 306] use

either PyBullet [44] or MuJoCo [273] as their underlying physics engines, which only supports rigid-body simulation. Recently, environments supporting soft-body manipulation, such as SAPIEN [306], TDW [72] and SoftGym [155] adopted Nvidia FleX [175] engine to simulate deformable objects and fluids. However, FleX only supports limited types of materials, and its underlying Position-Based Dynamics (PBD) solver has been considered not physically accurate enough to produce faithful interactions between solids and fluids. (See Section 4.3 for a comparison).

Differentiable physics simulation Machine Learning with differentiable physics has gained increasing attention recently. One line of research focuses on learning physics models with neural networks [145, 210], which are intrinsically differentiable and could be used for downstream planning and optimization. Learning these models assumes ground-truth physics data provided by simulators, hasn't proved to be capable of simulating a wide range of materials, and could suffer from out-of-domain generalization gap [235]. Another approach is to implement physics simulations in a differentiable manner, usually with automatic differentiation tools [48, 101, 108, 309], and the gradient information provided by these simulations has shown to be helpful in control and manipulation tasks concerning rigid and soft bodies [108, 143, 156, 308, 309]. On a related note, Taichi [102] and Nvidia Warp [174] are two recently developed domain-specific programming language for GPU-accelerated simulation. FluidLab is implemented using Taichi.

4.3 FluidEngine

4.3.1 System Overview

FluidEngine is developed in Python and Taichi, and provides a set of user-friendly APIs for building simulation environments. At a higher level, it follows the standard OpenAI Gym API and is compatible with standard reinforcement learning and optimization algorithms. Each environment created using FluidEngine consists of five components: i) a robotic agent equipped with user-defined end-effector(s); ii) objects imported from external meshes and represented as signed distance fields (SDFs); iii) objects created either using shape primitives or external meshes, represented as particles; iv) gas fields (including a velocity field and other advected quantity

fields such as smoke density and temperature) for simulating gaseous phenomena on Eulerian grids; and v) a set of user-defined geometric boundaries in support of sparse computations. After adding these components, a complete environment is built by calling a `build` method, which enables inter-component communications for simulation. (See Appendix 4.9 for a sample code of a typically environment constructed following this pipeline). Our full source code will be released upon publication. We encourage readers to refer to our project page for visualizations of a diverse set of scenes simulated with FluidEngine.

4.3.2 Material Models

FluidEngine supports various types of materials, mainly falling under three categories: solid, liquid and gas. For solid materials, we support rigid, elastic and plastic objects. For liquid, we support both Newtonian and non-Newtonian liquids. Both solid and liquid materials are represented as particles, simulated using the Moving Least Squares Material Point Method (MLS-MPM) [100], which is a hybrid Lagrangian-Eulerian method that uses particles and grids to model continuum materials. For gas such as smoke or air, we simulate them as incompressible fluid on a Cartersian grid using the advection-projection scheme [259].

Elastic, plastic, and liquid Both elastic and plastic materials are simulated following the original MLS-MPM formulation [100]. For both materials, we compute the updated deformation gradient, internal force, and material stress in the particle-to-grid (P2G) step, and then enforce boundary conditions in grid operations. Corotated constitutive models are used for both materials. For plastic material, we additionally adopt the Box yield criterion [260] to update deformations. We refer readers to the original MPM papers [100, 260] for more details. Both viscous and non-viscous liquids are also modeled using the corotated models, and we additionally reset the deformation gradients to diagonal matrices at each step. For non-Newtonian liquids such as ice cream, we model them using the elastoplastic continuum model with its plastic component treatsd by the von Mises yield criterion (following Huang et al. [108]).

Rigid body Two types of geometric representations are used to support rigid bodies in FluidEngine. The first type is represented by particles. We simulate rigid objects by treating them as elastic objects first using MLS-MPM, then we compute an

object-level transformation matrix and enforce the rigidity constraint. Another type of rigid object is represented using signed distance fields (SDFs), usually generated by importing external meshes. This representation is more computation-efficient since it computes its dynamics as a whole, generally used for meshes with big sizes.

Gas We develop a grid-based incompressible fluid simulator following [259] to model gaseous phenomena such as air and smoke. A particle-based method is not well suited in these cases due to its (weakly) compressible limit.

Inter-material coupling MPM naturally supports coupling between different materials and particles, computed in the grid operations. For interactions (collision contact) between mesh-based objects and particle-based materials, we represent meshes as time-varying SDFs and model the contact by computing surface normals of SDFs and applying Coulomb friction [260]. For interaction between mesh-based objects and the gas field, we treat grid cells occupied by meshes as the boundaries of the gas field and impose boundary conditions in the grid operations. Coupling between gas and particle-based materials is also implemented at the grid level, by treating grid cells occupied by particles as boundaries.

4.3.3 Differentiability and Rendering

FluidEngine is implemented based on Taichi's auto diff system to compute gradients for simple operations. We further implement our own gradient computations for complex operations such as SVD and the projection step in the gas simulation, and efficient gradient-checkpointing to allow gradient flow over unlimited time horizons, unconstrained by GPU memory size. FludiEngine also comes with a real-time OpenGL-based renderer. More details are in Appendix 4.10.

4.3.4 Comparison with other simulation environments

There exist many simulation engines and environments for robotics research, and here we compare FluidLab and FluidEngine with some popular ones among them, including PyBullet [44], MuJoCo [273], FleX [175], SoftGym [155], PlasticineLab [108], SAPIEN [306], TDW [72], DiffSim [215], DiSECt [94], PhiFlow [99], JAX-Fluids [15] and DEDO [9]. Table 4.1 shows a comparison in terms of differentiability and supported material types. FluidLab covers a wider range of materials compared

Simulators	Differentiable	Solid			Liquid		Gas
		Rigid	Elastic	Plastic	Newt.	Non-Newt.	Air / Smoke
PyBullet		✓	✓				
MuJoCo		✓					
FleX		✓	✓		✓		✓[B]
SoftGym		✓	✓		✓		
PlasticineLab	✓[A]	✓		✓			
SAPIEN		✓	✓		✓		
TDW		✓	✓				
DEDO		✓	✓				
DiffSim	✓	✓	✓	✓			
DiSECt	✓	✓	✓	✓			
PhiFlow	✓				✓		✓
JAX-Fluids	✓				✓		✓
FluidLab	✓	✓	✓	✓	✓	✓	✓

Table 4.1: **Comparison with other popular simulators.** [A]Plasticine lab offers differentiable simulation, but doesn't provide gradient checkpointing, hence only supporting gradient flow over relatively short horizons. [B]FleX demonstrates the capability of smoke simulation in their demo video, but this feature and smoke rendering are not available in their released codebase.

to existing environments. In addition, many of these environments rely on physics engines written in C++, and some are not open-sourced (such as FleX), making interpretability and extensibility a major hurdle, while FluidLab is developed in Python and Taichi, enjoying both GPU-acceleration and user-extensibility.

4.3.5 Comparison with other differentiable simulators

SPNets

SPNets is a pioneering work in robotic manipulation of fluids. It is essentially a fluid simulator implemented as neural network layers, which makes it differentiable. Therefore, it is more like a differentiable simulator itself rather than a policy learning method. The parameters of SPNets are the physical parameters of the fluids. One limitation of SPNets is that its simulation capability: it only supports simulating single-phase non-Newtonian fluids, and only supports one-way coupling from static boundaries to the fluids, i.e. it doesn't support computing effects of fluids on other objects. In this work, one of our key insights is that in order to study realistic

fluid manipulation problems that are motivated by real-world scenarios with realistic complexity, it's crucial to be able to support a wide range of materials and their inter-coupling. Our motivation is to study problems beyond the single-phase water tasks proposed in SPNets, such as single-phase water pouring and catching, where only the effect of manipulators on the water is considered. Therefore, on the simulator side, SPNets is not able to simulate most of our propose tasks. In addition, on the task solving side, in the SPNets paper the author did propose to use gradients provided by the differentiable simulation to optimize trajectories for liquid control. Which is similar to our differentible physics based optimization method (DP-H), except that we proposed a few optimization techniques (DP) to help solving the tasks.

DiffSim

DiffSim is a differentiable simulation platform that leverages mesh-based representation and implicit differentiation, which bring faster computation and less memory cost. However, mesh-based representation is not able to simulate highly deformable materials such as fluids. In contrast to them, we opt for a particle-based representation since our work focuses on simulating fluid systems and relevant manipulation skills. We use MPM-based method which naturally supports particle-based materials ranging from solid to fluids, and also the coupling between them.

DiSECt

DiSECt is a differentiable tool tailored towards robotic cutting of soft materials. It also opts for a mesh-based representation and FEM to simulate dynamics, which is ideal for computing physical phenomena in cutting scenarios, such as plastic deformation, fracture, and crack propagation. Our work differs from them in that FluidEngine is a general-purpose simulator, and our focus is on manipulation scenarios involving different types of fluids. Therefore, although our simulator does not specialize in special cutting scenarios, it can support a wide range of daily manipulation tasks involving interaction between various material types, ranging from solid, liquid to gas.

Taichi and DiffTaichi

Taichi is a domain-specific programming language that provides GPU-accelerated parallel computation, and we use it to implement FluidEngine and FluidLab. DiffTaichi is the automatic differentiation system of Taichi that supports gradient computation of numerical operations coded using Taichi. On the other hand, Taichi and DiffTaichi come with many example codes of simulating different scenarios, but they are scene-specific simulation examples and there's a big gap between them and an actual simulation environment that can be used for repeatable robotics research. We list a number of major difference here. 1) DiffTaichi doesn't support building computation graph and gradient checkpointing, which is necessary for computing and storing gradient flow over long-horizon manipulation tasks, especially when GPU memory is a major bottleneck. Therefore, unlike in PyTorch, any computation step involving gradient flow needs manually setting up the computation graph, and our framework handles that. 2) Taichi does provide MPM-based simulation examples, but they don't support functions such as mesh loading, SDF computation, SDF-based collision detection, friction modelling, mesh to particle conversion, etc. 3) Taichi doesn't support grid-based 3D smoke simulation, and doesn't support coupling between smoke, liquid, and mesh-based rigid objects, which is precisely one of the contributions of our platform, since such coupling is necessary for studying realistic fluid manipulation problem. 4) DiffTaichi only provides gradient computation of simple operations, not supporting gradients for e.g. SVD computations and the projection step in gas field simulation. 5) FluidEngine provides a unified configuration system and interface for building custom environments and specifying robotic agents, and also provides function for storing and accessing gradients of the intermediate states and robot actions. 6) The examples provided with Taichi typically requires saving scene configurations to a file and using an external rendering software such as Houdini for visualization, while we implement a stand alone OpenGL-based renderer that runs in real time and provides easy visualization and also capability for developing image-based policy learning method. 7) FluidLab is a set of standardized manipulation tasks for developing and evaluating robot learning algorithms, built on top of FluidEngine. This is similar to SoftGym versus FleX. In summary, FluidEngine is a piece of complete software with various APIs that can be used easily for building up robotic research

environments, just like what people are able to do with PyBullet or Mujoco, except that FluidEngine additionally supports deformable and fluid materials, while also providing gradient information.

PhiFlow

PhiFlow is a differentiable simulation toolbox. However, the major functionality is a underlying differentiable PDE solver, which can be used to protype small scenes with fluids but is not capable of building up realisitc 3D robotic manipulation scenarios. It requires manually specifying boundary conditions for fluid simulation, which is not realistic if we want to use actual object meshes for manipulation. Also, it relies on grid-based simulation for fluids, while our particle based simulation is more ideal for simulating liquids, which does not require boundary specifying and supports sparse computation. Moreover, PhiFlow is a special-purpose simulator specific to flow simulation, and doesn't support other material types such as solids, which is important for conducting robotics research in manipulation.

JAX-Fluids

Similar to PhiFlow, JAX-Fluids is a special-purpose differentiable solver tailored to flow simulation. Therefore, it shares the similar limitations of PhiFlow as discussed above: it's not designed for general purpose multi-material simulation, not supporting dynamics of other materials such as rigid and plastic objects, and does not support features such as mesh processing, scene building, and realistic rendering. In contrast, FluidEngine is a full-stack simulation platform providing such features and useful for conducting robotic manipulation research.

4.4 Validation Experiments for FluidEngine

In order to validate the accuracy and reliability of FluidEngine, we conducted a number of validation simulations. One useful way of validating the correctness of simulation engines is to run simulation experiments and verify if the produced simulation results match known physical phenomena. Below we list a number of classical physical phenomena that are typically observable in rigid body and computational fluid

dynamics. We testified our simulation engine by running simulations to see if it can successfully reproduce the listed behaviors. Visual results of such simulations are included in our project website.

- Kármán vortex street: When a flow of fluid is separated by blunt bodies, a repeating pattern of swirling vortices would appear due to vortex shedding. We simulate a flow of fluid flowing through a rigid cylinder and visualize the top-down view. We colorized the fluid around the cylinder, and it can be seen that our simulator can faithfully produce a street of vortices.
- Magnus effect: This is an observable phenomenon commonly associated with a spinning object moving through a fluid. The path of the spinning object is deflected in a manner not present when the object is not spinning. One example of such phenomenon seen in daily life is the screw shot in football playing. This is an observable phenomenon commonly associated with a spinning object moving through a fluid. The path of the spinning object is deflected in a manner not present when the object is not spinning. One example of such phenomenon seen in daily life is the screw shot in football playing. Here we simulate a spinning ball in a fluid body. It can be observed that the spinning motion produces a deflection in the motion trajectory of the ball, and the extent of the deflection is dependent on its spinning velocity.
- Buoyancy: We verify here if objects with different density in a liquid body would incur different magnitude of buoyancy. It can be observed that the red all would float on the water surface due to buoyancy.
- Incompressibility and stable volume: Liquids are generally considered incompressible. Our MPM-based engine simulates weakly-compressible liquids, where the momentum and mass are updated every step via consecutive grid-particle-grid information passes. Therefore, one concern is if the simulated fluid bodies can maintain consistent pressure and volume through long-horizon simulations, without momentum and pressure loss. We verified our simulator can simulate a stable fluid body that maintains steady volume and pressure after 10,000 simulation steps.
- Conservation of momentum: When objects are in collision, their motions obey conservation of momentum. Here we test the collision of two objects with

exactly the same mass on a frictionless floor, where it can be observed that they exchanged their momentum after the collision as expected.

- Varying bouncing behaviors with varying plasticity: Here we show different behaviors of the object bouncing on the floor when we vary the plasticity of the object. From left to right, we increase the plasticity of the object. Our simulator can simulate bouncing behavior with varying plasticity, where it can be observed energy dissipates faster with a greater plasticity.
- Rayleigh–Taylor instability: This phenomenon describes the instability of an interface between two fluids with different densities, which occurs when the heavy component is above and pushing the light one. Here we simulate the behavior of two layers of fluids with different densities, where it can be observed that a plume of heavy fluid with small vortices emerges and evolves due to the interfacial instabilities.
- Dam break: This is a classical engineering test case, where a fluid volume falls due to gravity and splash within the domain. We successfully reproduce this phenomenon with our simulation engine.

Our simulators successfully reproduced these physical phenomena, as we show on our project site. We believe running such validation experiments could informatively help demonstrate the accuracy and reliability of our simulation engine.

4.5 FluidLab Manipulation Tasks

FluidLab presents a collection of complex fluid manipulation tasks with different fluid materials and their interactions with other objects. In general, we observe that manipulations tasks involving fluids can be categorized into two main groups: i) where fluid functions as a *transmitting media* and is used to manipulate other objects which are otherwise unreachable, and ii) where fluid is the *manipulation target* and the goal is to move or deform the fluid volume to a desired configuration. We present a suite of 10 different tasks in FluidLab, as shown in Figure 8.6, where in each task an agent needs to manipulate or interact with the fluids in the scene to reach a specific goal configuration, falling under the aforementioned two categories.

	Material type	Contact with manipulator	Role of fluids	Key challenge / unique characterist
Latte Art (Pouring)	Inviscid and viscous liquid	Indirect	Manipulation target	Coupling between different liquids
Latte Art (Stirring)	Inviscid and viscous liquid	Direct	Manipulation target	Coupling between liquids and the rigid ma
Ice Cream (Static)	Non-Newtonion fluid	Indirect	Manipulation target	Fine grained shape matching of non-Newto
Ice Cream (Dynamic)	Non-Newtonion fluid	Direct	Manipulation target	Additional interaction with the rigid man
Transporting	Inviscid liquid and rigid solid	Indirect	Transmitting medium	Use liquid to indirectly manipulate distan
Mixing	Inviscid and viscous liquid	Direct	Manipulation target	Coupling between different liquid bo
Pouring	Inviscid liquid	Direct	Manipulation target	Fine-grained goal configuration
Gathering (Easy)	liquid and elastic solid	Direct	Transmitting medium	Use liquid to indirectly manipulate distan
Gathering (Hard)	liquid and elastic solid	Direct	Transmitting medium	More complex environment configura
Air Circulation	Gas	Indirect	Manipulation target	Involving dynamics of highly unstable a

Table 4.2: Comparison and characteristics of our selected tasks.

4.5.1 Discussion on Task Selections

When designing our tasks, we would like them to cover a wide range of variations, and these are the main factors that we consider: 1) varying material type and their coupling, from inviscid liquid, to viscous liquid, to non-newtonian fluid, to gaseous fluid, 2) the contact between the robot manipulator and the fluid body (direct contact or indirect emitting), and 3) the role of fluids, as a transmitting medium or the manipulation target. Therefore, we came up with the 10 selected tasks. In Table 4.2, we compare the selected tasks over the above 3 factors, and identify the unique characteristics of each task. We believe our proposed suite of task selections covers a wide range of fluid manipulation tasks encountered in our daily life, and should be useful as a set of standardized tasks for studying fluid manipulation problem.

4.5.2 Details of Evaluation Tasks

FluidLab contains 10 different manipulation tasks presented in Figure 8.6, all of which are motivated by real-world scenarios where an agent interacts with certain types of fluids. We additionally discuss the our reasoning behind the choices of our task selections in Appendix 4.5.1.

Fined Grained Pouring A glass filled with two types of non-Newtonian liquid with different densities, colored as blue and white respectively. The task is to pour the light blue liquid out of the glass while retaining as much white heavy liquid as possible.

Gathering (Easy & Hard) An agent equipped with a rigid plate is able to paddle the water in a tank, but its motion is restricted within the green area. The goal is to make water currents so that the two floating objects will be gathered towards

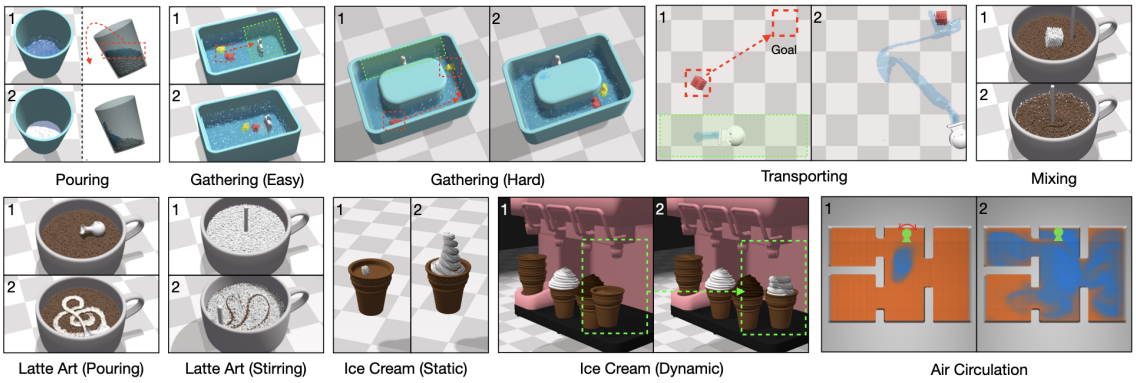


Figure 4.2: 10 Fluid manipulation tasks proposed in FluidLab. In each task, 1 depicts the initial condition and 2 shows a desired goal configuration. Both top and side views are shown for Pouring.

the marked goal location. The easy setting has a relatively open space while in the hard setting, the spatial configuration is more complex and the two objects need to move in different directions.

Transporting A water jet bot is required to move an object of interest to a randomly sampled goal location. The agent is only allowed to move within the green area, and thus can only transport the object using the water it injects.

Mixing Given a cube of sugar put in a cup of coffee, the agent needs to plan an sequence of stirring action and mixes the sugar particles with the coffee in the shortest period of time.

Latte Art (Pouring & Stirring) We consider two types of latte art making tasks: the first one is making latte art via pouring, where an agent pours milk into a cup of coffee to form a certain pattern; the second one is via stirring, where a layer of viscous frothed milk covers a cup of coffee, and the agent needs to manipulate a rigid stick to stir in the cup to reach a specified goal pattern. Note that due to the nonlinear fluid dynamics governing the system, these two tasks are much more complex than simple drawing on a static canvas. For example, during pouring, where a milk particle ends up being at the end of the task may be far away from where it was originally poured from, due to continuous interaction with the coffee beneath it and motion of the fluid body caused by pouring.

Ice Cream (Static & Dynamic) This task involves policy learning concerning a non-Newtonian fluid – ice cream. We consider two settings here. The first one is a

relatively simple and static setting, where we control an ice cream emitter that moves freely in space and emits ice cream into a cone and form a desired spiral shape. The second setting is more dynamic, where ice cream is squeezed out from a machine and an agent controls the biscuit cone. The goal configuration is defined with respect to the cone itself. During the process, the agent needs to reason about the dynamics of the icecream and its interaction with the cone since the ice cream moves together with the cone due to friction.

Air Circulation The agent controls the orientation of an air conditioner (denoted in green in the figure) which generates cool air (colored in blue) in a house with multiple rooms, initially filled with warm air (colored in orange), and the goal is to cool down only the upper left and the right room, while ensuring the lower left room maintains at its original temperature. Completing this challenging task requires fine-grained control and reasoning over the motion of the cooling air.

4.5.3 Task and Action representations

We model each task as an finite-horizon Markov Decision Process (MDP). An MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and a reward function associated with each transition step $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The transition function is determined by the physics simulation in FluidLab. We discuss the loss and reward for each task used for experiments in Section 8.5. The goal of the agent in the environment is to find a policy $\pi(a|s)$ that produces an action given the current state, and maximize the expected total return $E_\pi[\Sigma_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t)]$, where $\gamma \in (0, 1)$ denotes the discount factor and T is the horizon of the environment.

State, Observation, and Action Space We represent MPM-based materials with a particle-based representation, using a $N_p \times d_p$ state matrix, where N_p is the number of all particles in the system, and $d_p = 6$ contains the position and velocity information of each particle. To represent the gas field, which is simulated using a grid-based representation, we use another $N_c \times d_c$ matrix to store information of all grid cells, where N_c is the number of cells in the grid, and d_c is a task-dependent vector. In the air circulation task, $d_c = 4$ contains the velocity (a 3-vector) and temperature (a scalar) of each cell. In addition, the state also includes the 6D pose and velocity (translational and rotational) information of the end-effector of the robotic agent in

the scene. All the tasks use an MPM grid of size 64^3, with around 100,000 particles, and a 128^3 grid for gas simulation. The full action space of FluidLab is a 6-vector, containing both translational and angular velocity of the agent's end-effector. For more details of our task settings, their simulations, and the input to RL policies, please refer to Appendix 4.11.1.

4.6 Experiments

We evaluate model-free RL algorithms, sampling-based optimization methods as well as trajectory optimization using differentiable physics coupled with our proposed optimization techniques in FluidLab. We discuss our optimization techniques, and result findings in this Section. For detailed loss and reward design, please refer to Appendix 4.11.2.

4.6.1 Techniques and Optimization Schemes Using Differentiable Physics

Differentiable simulation provides useful gradient information that can be leveraged for policy optimization. However, oftentimes the dynamics function could both be highly non-smooth (e.g. in case of a contact collision, where the gradient is hard to obtain), and exhibit a complex optimization landscape where the optimizer can easily get trapped in local minima. We discuss a few techniques and optimization schemes used in our experiments here for obtaining more informative gradients and leveraging them more effectively in the context of fluid manipulation.

Soft Contact Model In our simulation, the contact between particle-based fluid bodies and SDF-based rigid objects occurs in the grid operations. Such contact induces drastic state changes during the classical MPM process. We adopt a softened contact model in our simulation: we consider contact occurs when the particles and the rigid objects are within a certain distance threshold ϵ, and compute an after-collision velocity v_c. Afterwards, we blend this velocity with the original velocity of the objects using a distance-based blending factor α, given by $v_{\text{new}} = \alpha v_c + (1-\alpha)v_{\text{original}}$, where $\alpha = \min\{\exp(-d), 1)\}$ and d is the distance between the two objects. This provides a smoothed contact zone and thus offers smoothed gradient information during contact.

Temporally Expanding Optimization Region During gradient-based trajectory optimization, if we consider the full loss computed using the whole trajectory with respect to a certain goal configuration, the whole optimization landscape could be highly nonconvex and results in unstable optimization. In our experiments, we warm start our optimization with a restricted temporal region of the trajectory and keeps expanding this region steadily whenever a stabilized optimization spot is reached for the current region. This greatly helps stabilize the optimization process using differentiable physics and produces better empirical results, as we show in Section 4.6.2.

Gradient Sharing within Fluid Bodies Many optimization scenarios involving fluids could be highly non-convex. Consider a simple pouring task where the goal is to pour all the water from a mug, with the loss defined as the distance between the ground floor and all the water particles. During the exploration phase, if a water particle was never poured out of the mug, it is trapped in a local minimum and the loss gradient won't be able to guide towards a successful pouring. Thanks to the continuity of fluid materials, we are able to share gradients between adjacent particles in a fluid body and use those informative gradients to guide other particles. In our experiments where fluid is required to reach a goal configuration, we enable gradient sharing by updating the gradient on each particle via accumulating neighboring particle gradients as follows: $\frac{\partial \mathcal{L}}{\partial q_i} = \Sigma_j w(\mathcal{L}_j, d_{ij}) \frac{\partial \mathcal{L}}{\partial q_j}$, where j iterates over all neighboring particles of i, q represents particle properties such as velocity or position, and w is a kernel function conditioned on both the distance between i and j and the loss of particle j, assigning a higher weight to neighboring particles who incurred a lower loss at the previous iteration.

4.6.2 Method Evaluation with FluidLab Tasks

We evaluate our proposed optimization schemes coupled with differentiable physics (DP), model-free RL algorithms including Soft Actor-Critic (SAC) [85] and Proximal Policy Optimization (PPO) [238], CMA-ES [89], a sampling-based trajectory optimization method, as well as PODS [186] an method combines RL and differentiable simulation. We use the same accumulated reward to evaluate all the methods. Additionally, we include the performance of an oracle policy controlled by a human

Task	Latte Art (Pour.)	LatteArt (Stir.)	Ice Cream (Sta.)	Ice Cream (Dyn.)	Transporting
SAC	-1417.6 ± 37.4	555.4 ± 22.8	-1450.7 ± 14.2	-509.6 ± 30.9	**2050.6 ± 450.7**
PPO	-1078.3 ± 187.4	585.3 ± 21.7	-1250.6 ± 563.8	-424.0 ± 110.5	1104.6 ± 99.7
CMA-ES	279.9 ± 124.1	745.2 ± 22.1	69.5 ± 140.1	131.4 ± 100.5	1694.5 ± 463.6
PODS	872.3 ± 17.6	454.6 ± 73.5	251.7 ± 43.1	-203.6 ± 78.1	**2120.2 ± 296.1**
DP-H	**998.7 ± 0.5**	830.1 ± 7.2	295.1 ± 14.2	-162.9 ± 30.5	1443.9 ± 78.6
DP	**999.3 ± 0.7**	**995.4.6 ± 1.68**	**474.6 ± 21.4**	**1469.4 ± 28.8**	1471.1 ± 127.7

Task	Mixing	Pouring	Gathering (Easy)	Gathering (Hard)	Air Circulation
SAC	1523.2 ± 389.4	374.5 ± 7.3	130.2 ± 2.4	477.2 ± 9.7	4430.2 ± 264.7
PPO	1728.3 ± 159.4	-324.3 ± 50.3	129.7 ± 1.2	-138.9 ± 67.3	4282.2 ± 610.9
CMA-ES	3875.2 ± 143.2	57.1 ± 169.7	143.1 ± 1.4	687.2 ± 80.1	3917.3 ± 385.3
PODS	4742.3 ± 187.6	284.1 ± 84.9	129.5 ± 1.7	413.4 ± 71.6	3557.8 ± 364.2
DP-H	5974.0 ± 198.8	88.3 ± 9.1	116.9 ± 0.3	566.9 ± 30.4	**5046.3 ± 23.7**
DP	**7648.4 ± 47.0**	**603.4 ± 29.7**	**151.5 ± 0.3**	**946.8 ± 5.1**	**5043.3 ± 21.2**

Table 4.3: The final accumulated reward and the standard deviation for each evaluated method.

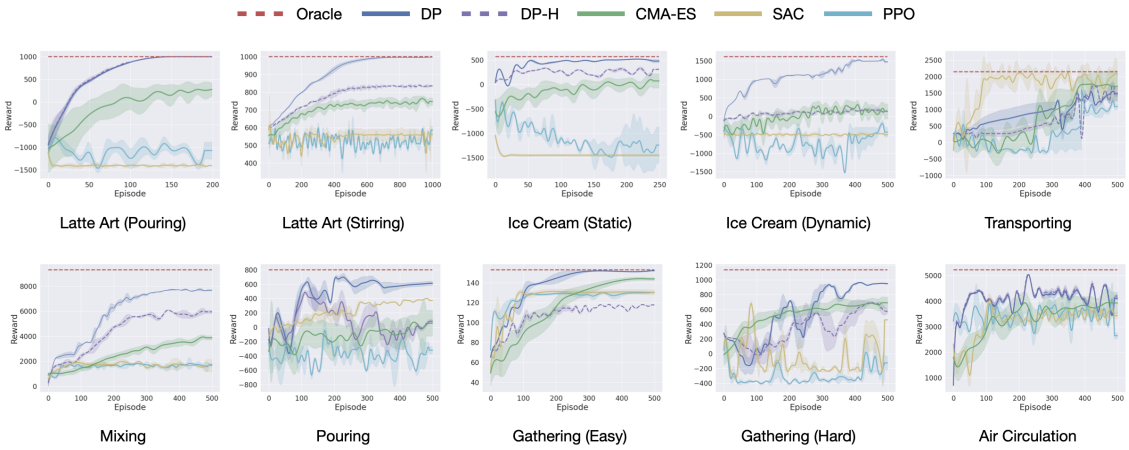


Figure 4.3: Reward learning curve for all evaluated methods in each task.

operator as a reference. The reward learning curves are shown in Appendix Figure 4.3, and we report the final performance of all methods upon convergence in Table 4.3.

Model-free RL Algorithms RL algorithms demonstrate reasonable learning behaviors in tasks with relatively small observation spaces, such as Transporting (where only a subset of water particles are present in the scene at each step) and Air Circulation. However, in environments such as Latte Art, Mixing, and Ice Cream,

RL methods struggle to generate a reasonable policy. We suspect that this is because 1) the high-dimensional space used to describe the fluid system, which is hard for RL algorithms to abstract away useful information about the system within limited exploration iterations and training budget, making it less sample efficient than DP, and 2) the non-linear dynamics of the complex fluids such as viscous frothed milk and the non-Newtonian ice creams, as well as their interactions with other solid materials, make policy search of RL policies based on sampled actions extremely difficult since randomly sampled actions and rewards are not informative enough in conveying contact and deformation information about the fluid systems.

Sampling-based trajectory optimization CMA-ES does reasonably well in many tasks with simple spatial configuration as goals. However, it struggles in fine-grained shape matching tasks (*e.g.* Ice Cream) and fine-grained control tasks (*e.g.* pouring). This is because CMA-ES searches for action sequences via trajectory sampling, and thus cannot handle delicate interactions with fluid materials well.

Differentiable physics-based trajectory optimization In most of the tasks, the gradients provided by our differentiable simulation are able to guide trajectory optimization towards a good optimization spot, resulting in near-oracle performance. It also demonstrates better sample-efficiency than other methods in most tasks. We additionally compare with a *hard* version of differentiable physics (DP-H), which uses the gradient to optimize trajectory, but without using the optimization techniques proposed in Section 4.6.1. DP-H does similarly well on tasks with relatively simple settings, but fails to produce good trajectories in scenarios with interactions between fluids and other materials bodies, as well as in tasks with more complex and non-convex optimization landscapes, such as Pouring with fine-grained control objective. The experimental results demonstrate that the gradient information provided by our differentiable simulator, coupled with the optimization schemes we proposed, provides informative optimization information when dealing with complex fluid systems. We additionally evaluate the optimized trajectories in a real-world robotic setup and show that the learned trajectories can perform the desired tasks reasonably well in the real world. See Appendix 4.7 for more details.

Comparison with PODS PODS is a method that combines differentiable simulation and RL. It performs well in relatively convex tasks such as Latte Art (Pouring), and even outperforms DP and DP-H on simple tasks like transporting,

mainly due to its exploratory nature from RL. However, it is not able to match performance in other complex tasks, especially in those where DP shows an clear advantage over DP-H and RL methods. We believe this is because PODS only uses local gradient information without our proposed optimization techniques; also, it is additionally learning a closed-loop policy which maps observation to action, where DP is only optimizing a trajectory. In fact, what PODS is learning is precisely the next step of our research, where a better scene representation to describe the fluid system is needed to better distill DP-optimized trajectory to a closed-loop policy network. This comparison suggests that a future research direction is how to encode the scene in a more informative representation for better policy learning, such as a pointcloud based representation; in addition, we expect better performance by combining our proposed optimization landscapes with methods that use both differentiable simulation and RL. These remain as our future work.

Potential future research directions In our experiments above, we evaluated a range of policy learning and trajectory optimization methods. Our results show that when coupled with our proposed optimization scheme, differentiable physics can provide useful gradient information to guide trajectory optimization, and show reasonable performance in a range of complex task settings, even in those with fine-grained shape matching goals and highly unstable fluid systems such as air flow. Meanwhile, our results suggested a number of major challenges in solving complex fluid manipulation methods. First of all, the model free RL methods and also the method combining RL and differentiable simulation for policy learning struggle to produce competitive results. We believe one main reason for this is it's difficult for the policy to informatively interpret the high-dimensional state of the complex fluid systems. It's worth investigating what would be an efficient representation for learning such closed-loop policies: point cloud-based representation or image input could be a good starting point, and more abstract representations with semantic understanding of the material properties and appearances are also worth studying. Second, one clear goal is to combine our optimization scheme with policy learning, and to distill optimized trajectories using differentiable physics into more general close-loop control policies. Thirdly, although our current task selections cover a range of different scenarios and materials, they are still relatively short-horizon, in the sense that there's usually only a single action mode. It would be very interesting to

extend task design to more realistic latte-art making, which requires action switching between both pouring and stirring. One promising direction is to learn a more abstract dynamics model of the fluid system and use it for long-horizon action planning and skill selection. Also, our current latte art tasks rely on dense reward signal computed from a temporal trajectory of goal patterns, and a more abstract and learned dynamics model would probably allow optimization using a static goal pattern. Last but not least, our real world experiments show that there's still a perceivable gap between our simulated material behavior and the real world environments. Improving both simulation performance and close-loop policy learning are needed to enable robots to perform better in more complex tasks, such as more realistic latte art making.

4.7 Sim-to-real Transfer

Our simulation platform provides an efficient test bed and useful gradient information for developing algorithms to deal with complex fluid manipulation problems, testing how the optimized policies perform in a real-world setting would be very helpful and informative in evaluating the sim-to-real gap and suggesting future research directions. Therefore, we conducted experiments on a real robot system, with a 7-DoF Franka Emika robot equipped with a parallel jaw gripper. We picked four representative tasks proposed in FluidLab, including Latte Art (Stirring), Ice Cream (Dynamic), Gathering and Mixing, and set up corresponding real world scenarios. We optimize the trajectories in simulation, and then command the robot to execute them using velocity control in the corresponding real-world scenarios. (See our project site for qualitative results.) It can be observed that although there's certain sim-to-real gap in terms of material behavior and dynamics, when we apply the simulation-optimized trajectories to real-world, the tasks can be completed to a reasonable extent. The sim-to-real gap mainly comes from the simulation inaccuracy and difference in material properties between sim and real. For example, in the latte art experiment, the frothed milk behaves a bit differently than in simulation: it's more sticky and tend to mix again after the latte art pen passes. This could be further improved in simulation by developing more accurate material models, as well as increasing the simulation resolution and number of particles, which we will strive to improve in our future work. Another example is the ice cream experiment, where it is difficult to maintain

a steady flow speed of the ice cream, resulting in slight different end shape of the ice cream compared to the simulated one. Although there exists such sim-to-real gap in material dynamics, the robot is able to complete the proposed tasks reasonably well. We would also like to acknowledge that given our current resources, we are not able to conduct all the proposed tasks in real world, such as the air circulation task, which remains as our future work. However, We believe the current 4 selected tasks cover a representative range of materials and task settings, and such experiments could shed light on the value of our simulator as a test bed for robotic research, point out potential sim-to-real gap and suggest possible improvements in our future work.

4.8 Conclusion

We presented FluidLab, a virtual environment that covers a versatile collection of complex fluid manipulation tasks inspired by real-world scenarios. We also introduced FluidEngine, a backend physics engine that can support simulating a wide range of materials with distinct properties, and also being fully differentiable. The proposed tasks enabled us to study manipulation problems with fluids comprehensively, and shed light on challenges of existing methods, and the potential of utilizing differentiable physics as an effective tool for trajectory optimization in the context of complex fluid manipulation. One future direction is to extend towards more realistic problem setups using vision as input. One potential way is to distill policy optimized using differentiable physics into neural-network based policies [156], or use gradients provided by differentiable simulation to guide policy learning [309]. We also plan to test policies trained using our simulation in real-world setups. Furthermore, since MPM was originally proposed, researchers have been using it to simulate various realistic materials with intricate properties, such as snow, sand, mud, and granular materials. These more sophisticated materials can be seamless integrated into the MPM simulation pipeline in FluidLab, and will be helpful for studying manipulation tasks dealing with more diverse set of materials. We believe the standardized task suites and the general purpose simulation capability of FluidLab will benefit future research in robotic manipulation involving fluids and other materials.

4.9 Sample code for building a simulation environment using FluidEngine

```
import os
import gym
import numpy as np
from gym.spaces import Box
from fluidlab.configs.macros import *
from fluidlab.engine.scenes import Scene
import fluidlab.utils.misc as misc_utils

class FluidEnv(gym.Env):
    '''
    Base env class.
    '''
    def __init__(self, version, loss=True, loss_type='diff', seed=
    None):
        if seed is not None:
            self.seed(seed)

        self.horizon               = 500
        self.horizon_action        = 500
        self.target_file           = None
        self._n_obs_ptcls_per_body = 200
        self.loss                  = loss
        self.loss_type             = loss_type
        self.action_range          = np.array([-1.0, 1.0])

        # create a simulation
        self.scene = Scene()
        self.build_env()
        self.gym_misc()

    def seed(self, seed):
        super(FluidEnv, self).seed(seed)
        misc_utils.set_random_seed(seed)

    def build_env(self):
```

```
35          self.setup_agent()
36          self.setup_statics()
37          self.setup_bodies()
38          self.setup_gas_field()
39          self.setup_boundary()
40          self.setup_renderer()
41          self.setup_loss()
42
43          self.scene.build()
44          self._init_state = self.scene.get_state()
45
46          print(f'===>  {type(self).__name__} built successfully.')
47
48      def setup_agent(self):
49          agent_cfg = CfgNode(new_allowed=True)
50          agent_cfg.merge_from_file(get_cfg_path('agent_scooping.yaml'
    ))
51          self.scene.setup_agent(agent_cfg)
52          self.agent = self.scene.agent
53
54      def setup_statics(self):
55          # add static mesh-based objects in the scene
56          self.scene.add_static(
57              file='tank.obj',
58              pos=(0.5, 0.4, 0.5),
59              euler=(0.0, 0.0, 0.0),
60              scale=(0.85, 0.92, 0.92),
61              material=TANK,
62              has_dynamics=False,
63          )
64
65      def setup_bodies(self):
66          # add fluid/object bodies
67          self.scene.add_body(
68              type='cube',
69              lower=(0.12, 0.25, 0.18),
70              upper=(0.88, 0.45, 0.82),
71              material=WATER,
72          )
```

```
        self.scene.add_body(
            type='mesh',
            file='duck.obj',
            pos=(0.5, 0.6, 0.55),
            scale=(0.10, 0.10, 0.10),
            euler=(0, -90.0, 0.0),
            color=(1.0, 1.0, 0.3, 1.0),
            filling='grid',
            material=ELASTIC,
        )
        self.scene.add_body(
            type='cylinder',
            center=(0.2, 0.5, 0.55),
            height=0.08,
            radius=0.02,
            euler=(0.0, 0.0, -45.0),
            color=(1.0, 0.5, 0.5, 1.0),
            filling='natural',
            material=ELASTIC,
        )

    def setup_gas_field(self):
        self.scene.setup_gas_field(
            res          = 128,
            dt           = 0.03,
            solver_iters = 50,
            decay        = 0.99,
            q_dim        = 1,
        )

    def setup_boundary(self):
        self.scene.setup_boundary(
            type='cube',
            lower=(0.12, 0.25, 0.18),
            upper=(0.88, 0.95, 0.82),
        )

    def setup_renderer(self):
        self.scene.setup_renderer()
```

```
112
113     def setup_loss(self):
114         pass
115
116     def gym_misc(self):
117         obs = self.reset()
118         self.observation_space = Box(DTYPE_NP(-np.inf), DTYPE_NP(np.
    inf), obs.shape, dtype=DTYPE_NP)
119         if self.scene.agent is not None:
120             self.action_space = Box(DTYPE_NP(self.action_range[0]),
    DTYPE_NP(self.action_range[1]), (self.scene.agent.action_dim,),
    dtype=DTYPE_NP)
121         else:
122             self.action_space = None
123
124     def reset(self):
125         self.scene.set_state(**self._init_state)
126         return self._get_obs()
127
128     def step(self, action):
129         self.scene.step(action)
130
131         obs    = self._get_obs()
132         reward = self._get_reward()
133
134         assert self.t <= self.horizon
135         if self.t == self.horizon:
136             done = True
137         else:
138             done = False
139
140         info = dict()
141         return obs, reward, done, info
142
143     def render(self, mode='human'):
144         assert mode in ['human', 'rgb_array']
145         return self.scene.render(mode)
```

4.10 Details on FluidLab's Differentibility and Rendering

Differentiability and gradient checkpointing We use the automatic differentiation tool (autodiff) provided by Taichi to get gradients for most of our computations; for more complex computations and kernels not supported by Taichi's autodiff, we implemeneted gradient computation ourselves, including SVD computations and the projection step in gas simulation. One additional challenge for making the whole simulation fully differentiable is enabling gradient flow through time, which requires bookkeeping system states of all timesteps. Since most of FluidLab's tasks have tens of thousands of simulation steps, it is impossible to fit the whole computation graph into the GPU memory. In order to address this, we implemented efficient gradient checkpointing to enable gradient flow over unlimited time horizons. During the forward process of the simulation, only a local trajectory of the environment states are stored in the GPU memory for fast computation. Whenever the GPU memory is full, we store the latest state of the environment to a caching device (either CPU memory or local disks), and clear up the GPU memory for further simulation. Once a complete forward pass is done, we restore the cached states and run forward simulation again to fulfill the GPU memory with the local chunk of simulation trajectory. Note that instead of caching all intermediate states, we only store one single step of the states at every caching step, and run simulation again to restore the states of the local trajectory; therefore, the i/o step of such caching operations is very efficient. This enables us to propagate loss and gradient information backward in time through a simulation trajectory with arbitrary number of steps, unconstrained by the GPU memory size.

Rendering We implemented a photo-realistic renderer for visualizing scenes created with FluidEngine. The renderer is developed in C++ using OpenGL, supports GPU-accelerated real-time rendering. Part of our rendering pipeline was modified based on FleX's rendering system, enhanced with various features such as particle-level colorization, headless rendering, dynamic object loading, smoke rendering, etc. We also provided a set of python APIs to dynamically update the scene configuration from within Python. The renderer supports rendering materials as either particles or

fluids. Particle-based rendering is more interpretable for visualization and debugging purposes, while fluid-based rendering is more photo-realistic and allows possible image-based sim-to-real transfer applications in the future. We present a comparison of these two rendering methods in Figure 4.4.

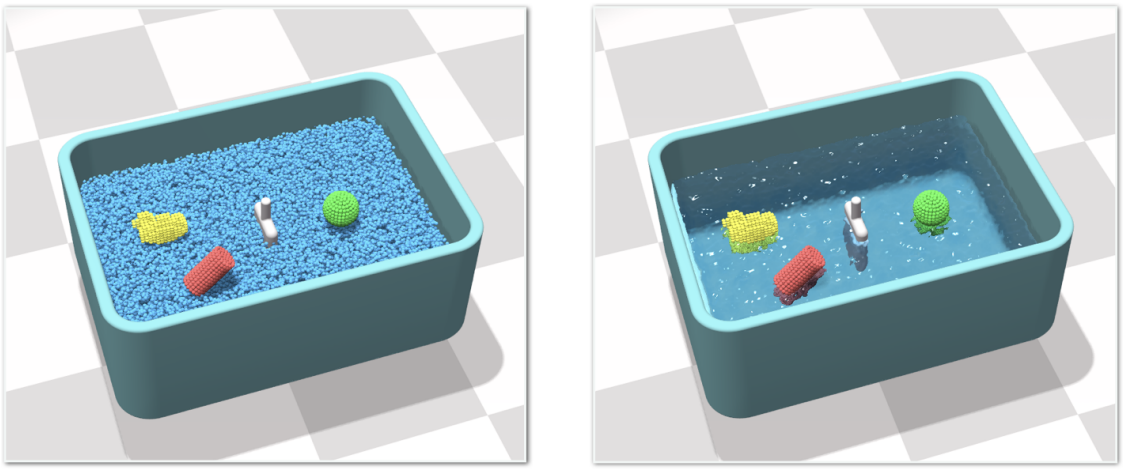


Figure 4.4: A rendered scene with water and floating objects. Left: water rendered as particles. Right: realistic water rendering.

4.11 Additional Details on FluidLab Tasks and Evaluations

4.11.1 Task Details

For RL algorithms, we down sample the number of particles and cells for each fluid body in the scene to avoid a intractably huge observation space. The state matrix is then flattened and fed into an MLP policy network. We do not evaluate image-based RL methods as many tasks FluidLab are 3D and occlusion could be a potential problem, and leave such evaluations for future work.

All our tasks use a simulation step of 2e-3 seconds, each containing 10 substeps of 1e-4 seconds for ensuring simulation stability. All our proposed tasks contain around 100,000 particles and runs in real time (60 FPS where each frame is a simulation

step) on a desktop computer equipped with an Nvidia RTX 3080 GPU and a Intel i7-8700K CPU, and the typical GPU usage is under 50%.

Each task has a task-specific action range and action space, with certain dimensions of the full action-space locked. We present our task-specific settings for action spaces, observed number of particles per fluid body and observed number of grid cells in Table 4.4. Detailed properties of all materials used are listed in Table 4.5.

Task	Action space		number of particles	number of grid cells
	Trans. Velocity	Rot. Velocity	per fluid body	for gas field
Latte Art (Pouring)	Along x and z	-	1000	-
Latte Art (Stirring)	Along x and z	-	1000	-
Ice Cream (Static)	Along x, y and z	-	2000	-
Ice Cream (Dynamic)	Along x, y and z	-	2000	-
Transporting	Along x and z	Around y	500	-
Mixing	Along x and z	-	1000	-
Pouring	-	Around z	500	-
Gathering (Easy)	Along x and z	-	500	-
Gathering (Hard)	Along x and z	-	500	-
Air Circulation	-	Around y	-	$12 \times 12 \times 12$

Table 4.4: Task-specfic settings.

4.11.2 Loss and Reward

Latte Art (Pouring), **Latte Art (Stirring)**, **Ice Cream (Static)** and **Ice Cream (Dynamic)**: These tasks have a set of predefined goal patterns, and the loss is defined by summing up the Chamfer distance between particles in the current state of the scene and the particles in the goal pattern at each time step. Ideally, the goal pattern should be a static goal shape of the final step, but using a single goal pattern would be extremely challenging for RL methods: if only the last step of the trajectory is used for reward computation, such an extremely sparse reward will make learning infeasible; computing a dense reward based on the distance between the current state of each timestep and the goal pattern will also be non-informative: due to

Material	Type	Lamé parameters		Density ρ
		μ	λ	
Water	Liquid	0.0	277.78	1.0
Milk	Liquid	0.0	277.78	1.0
Frothed Milk	Viscous Liquid	208.33	277.78	1.0
Coffee	Liquid	0.0	277.78	1.0
Ice Cream	Non-newtonian Liquid	416.67	277.78	0.5
Floating object	Elastic	416.67	277.78	0.5
Sugar cube	Viscous Liquid	208.33	277.78	1.0
Light Liquid in Pouring	Liquid	0.0	277.78	0.8
Heavy Liquid in Pouring	Liquid	0.0	277.78	1.5
Object in Transporting	Rigid	416.67	277.78	5.0

Table 4.5: Details of materials properties used in FluidLab.

the dynamic movement within the fluid bodies, mimicing the static goal pattern at each timestep will not result in successful learning. Therefore, in order to make an informative comparison, we use a temporal trajectory of the desired goal pattern to compute the loss $L = \Sigma_t \text{CD}(s_t, s_t^{goal})$, where CD denotes Chamfer Distance.

Transporting, **Pouring**, **Gathering (Easy)** and **Gathering (Hard)**: These tasks use a single spatial location as the goal and the loss is computed as: $\mathcal{L} = \Sigma_i \|p_i - p_{goal}\|$, which is summed over all particles of the object(s) to be manipulated. For **Pouring**, we apply this loss between the particles of the upper layer fluid and the ground location, with an additional loss attracting the particles of the lower layer fluid to their initial positions.

Mixing The goal of this task is to mix the sugar particles with the coffee. The loss is designed to maximize the spatial coverage of the sugar particles and is defined as $\mathcal{L} = -\Sigma_i \Sigma_j \|p_i - p_j\|$, where both i and j iterate over all particles originally belonging to the sugar cube.

Air circulation There are 27 sensors placed in the house, with 9 sensors in each room uniformly. The loss is defined as $\mathcal{L} = \Sigma_{\text{ul}} |T_{\text{ul}} - T_{\text{cool}}| + \Sigma_{\text{r}} |T_{\text{r}} - T_{cool}| + \Sigma_{\text{ll}} |T_{\text{ll}} - T_{\text{warm}}|$, where T represents temperature, and ul, r and ll denote sensors in the upper

left, right and lower left room respectively.

We use the losses discussed above for optimization-based methods; for model-free RL methods, the reward of each task is simply defined as $\mathcal{R} = c_1 - c_2\mathcal{L}$, where c_1 and c_2 are constants to ensure a reasonable reward scaling for each task.

Chapter 5

Genesis: A Universal and Generative Physics Engine

5.1 Introduction

Simulation has played an crucial role in robotics research, providing a solid foundation for training robotic policies and generating data, leveraging the ever increasing power of computation. However, robotics researchers have long been limited by usability issues and opaqueness of available simulators. Existing GPU-accelerated simulators often come with steep learning curves due to intricate, data-centric concepts, complex APIs, and complicated software stacks — making mastering them a daunting task for researchers, especially for newcomers in the field. Furthermore, some simulators are closed-source, restricting transparency and limiting researchers' ability to understand, control, or iteratively improve the underlying physics solvers based on e.g. real-world observations and feedback.

Therefore, we launched the *Genesis* collaborative effort since Jan 2022, by uniting robotics and computer graphics researchers and engineers from near twenty institutes and industrial labs to work together on a next-generation robotic simulation platform. This effort was originally motivated by FluidLab [305] (Chapter 4) and several other relevant works including RoboNinja [313], SoftZoo [287], etc, which implemented differentiable physics engine for simulating deformable objects to support research

projects involving robotic manipulation and locomotion. Our vision is to build a fully transparent, user-friendly ecosystem where contributors from both physics simulation and robotics backgrounds can come together to collaboratively create a high-efficiency, realistic (both physically and visually) virtual world for robotics research and beyond. In addition, we recognize the wealth of innovative algorithms that are continuously developed by the computer graphics community for both simulation and rendering; however, there hasn't been a collaborative effort so far that attempts to bring together these algorithms to create a realistic and compute-powered virtual realm for embodied and physical intelligence to emerge. The Genesis effort aims to bring the best of computer graphics technology into the robotics community, and build realistic virtual worlds for robots to explore and evolve.

This is still an on-going project and will be publicly release very soon. In this chapter, we present a brief technical report to introduce the whole Genesis system.

5.2 A Unified Framework for Multiple Physics Solvers

Various physics-based solvers tailored to different materials and needs have been developed in the past decades, mostly by the computer graphics community. Some prioritize high simulation fidelity, while others favor performance, albeit sometimes sacrificing accuracy. In fact, there doesn't exist a single solver that could support flexibly a wide distribution of material properties: for simulation robotic arms and legged robots, we need to approximate each link as a rigid body and connect them via simplified joints to reduce simulation complexity for optimal simulation performance. For tasks involving cutting doughs and pouring liquids, we need to simulate these high-dimensional objects using non-rigid representations such as particles or grids. For manipulation settings such as wearing cloths, we need mesh-based representation that simulates deformable cloth, while preserving its topology.

Genesis, in contrast to existing simulation platforms, natively supports a wide range of different physics solvers. We present a sketch of Genesis's software architecture in Figure 5.1. We abstract out 3 layers of information for creating an *entity* in a Genesis environment:

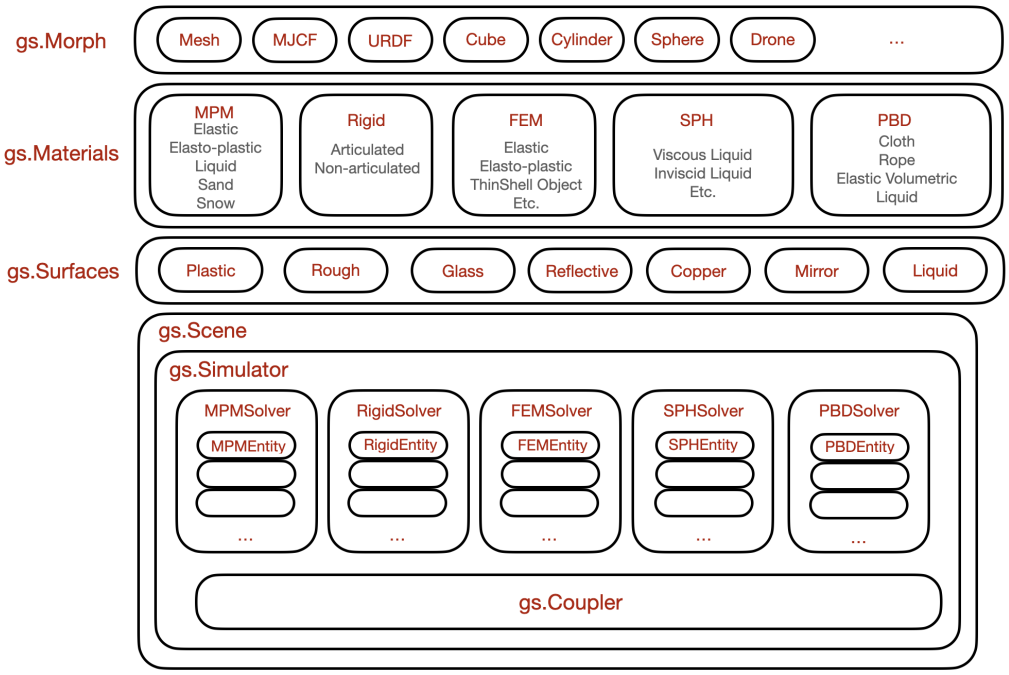


Figure 5.1: Genesis Architecture

- `gs.Morph` specifies morphology related information for creating an entity. This could be initiated from an external mesh file, a robot configuration file (such as `*.MJCF` or `*.URDF`, a shape primitive, or a drone configuration file.
- `gs.Material` sets the physical material of the entity. Note that this material setting determines which backend-solver will be invoked to simulate the created entity. We built a unified physics engine from the ground up that integrates solvers include: Material Point Method (MPM), Finite Element Method (FEM), Position Based Dynamics (PBD), Smoothed-Particle Hydrodynamics (SPH) and Articulated Body Algorithm (ABA)-based Rigid Body Dynamics.
- `gs.Surface` sets all rendering-related visual properties, and provides a set of pre-defined surface types, covering plastic, transparent, and metallic surfaces.

Note that due to this structured way of composing entities, users are able to easily change the physical property of a loaded entity, by simply switching the `gs.Material` attribute. All these entities are added to a `gs.Scene` environment, which wraps a `gs.Simulator` object, which manages all the underlying solvers. In addition, we

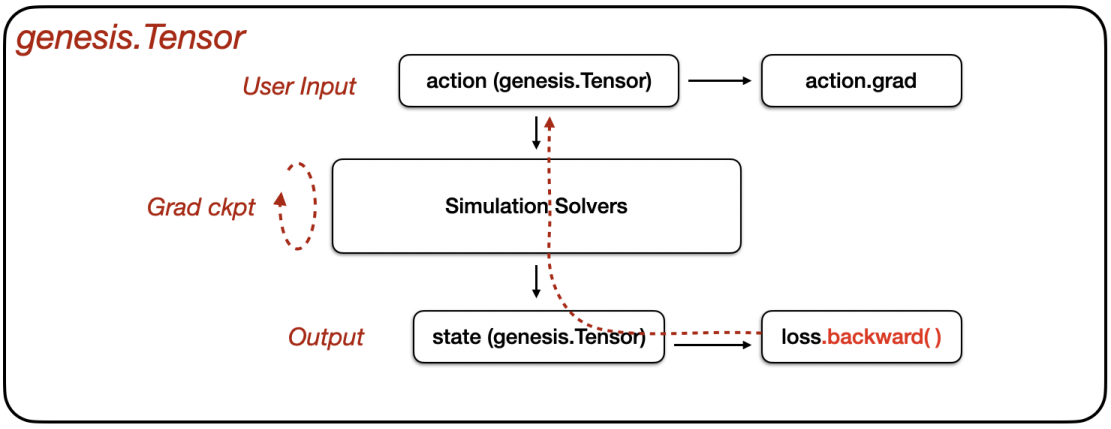


Figure 5.2: Genesis Architecture

have a simulator-level inter-solver coupling manager (`gs.Coupler`). During each simulation step, all the solvers first advect their own state, and then the coupler will handle inter-solver effects and compute the resulted force to simulate coupling effects between entities made of different materials.

Genesis is developped using Taichi [102], thus enjoy the speed of massive parallel simulation powered by GPU acceleration.

5.3 Differentiable Simulation

During our prior work [287, 305, 313], we observed that gradient information of the simulation is a privileged knowledge that could potentially help speed up skill training, especially for fine-grained tasks. Therefore, we implemented Genesis in a way that's fully compatible with differentiable simulation, and will make it fully differentiable in the long term. Differentiable simulation software oftentimes have a very intricate user interface and imposes a lot of restrictions for user to comply with and to access the gradient information. Genesis implements a custom Tensor system (`gs.Tensor`) seamlessly integrated with PyTorch. This integration guarantees that the Genesis simulation pipeline mirrors the operation of a standard PyTorch neural network in both functionality and familiarity. Provided that input variables are Genesis.Tensor, users are able to compute custom loss using simulation outputs, and one simple call of `loss.backward()` will trigger gradient flow all the way back through time back

to input variables, and optionally back to upstream torch-based policy networks, allowing effortless gradient-based policy optimization and gradient-accelerated RL policy learning, as shown in Figure 5.2.

5.4 Physics-Based Tactile Sensing

In the goal of enabling robots to perform human-level manipulation on a diverse set of tasks, touch is one of the most prominent components. Tactile sensing, as a modality, is unique in the sense that it provides accurate, fine-detailed information about environmental interactions in the form of contact geometries and forces. Although its efficacy has been highlighted by prior research, providing crucial feedback in grasping fragile objects [109], enabling robots to perform in occluded environment [321], and detecting incipient slip [34] for highly reactive grasping, there are still advances in tactile sensing to be made especially in the form of simulation. Existing robotic simulators either lack simulation for tactile sensing or limit interactions to rigid bodies. To accurately simulate tactile sensors which are inherently soft, it is essential to model soft body interaction's contact geometries, forces, and dynamics. Prior work [250] attempted to simulate contact geometries and forces for tactile sensors under (quasi-)static scenarios, and it was successfully applied to robotic perception tasks such as object shape estimation [266], and grasp stability prediction [251]. However, highly dynamic manipulation tasks have not been thoroughly explored. Other prior works approach contact dynamics by either approximating sensor surface deformation using rigid-body dynamics [310] or using physics-based soft-body simulation methods such as Finite Element Method (FEM) [194]. However, these methods are still limited to manipulating rigid objects.

To address existing limitations in simulating tactile sensing modules, Genesis presents DiffTactile, a physics-based differentiable tactile simulation system designed to enhance robotic manipulation with dense and physically accurate tactile feedback. In contrast to prior tactile simulators which primarily focus on manipulating rigid bodies and often rely on simplified approximations to model stress and deformations of materials in contact, DiffTactile emphasizes physics-based contact modeling with high fidelity, supporting simulations of diverse contact modes and interactions with objects possessing a wide range of material properties. DiffTactile is powered by

the physics solvers discussed above, and is implemented in a fully-differentiable manner. Its differentiable nature facilitates gradient-based optimization for both 1) refining physical properties in simulation using real-world data, hence narrowing the sim-to-real gap and 2) efficient learning of tactile-assisted grasping and contact-rich manipulation skills. Additionally, we introduce a method to infer the optical response of our tactile sensor by modeling the surface of the sensor as a height function, and represent the continuous spatially-varying reflectance function of the surface as a 4D vector-valued function. We then approximate our reflectance function with a neural module. DiffTactile has been released as a standalone research paper. For more details, we refer readers to the original paper [252].

5.5 Generative Simulation

As we will discuss in the next part, one goal of the Genesis effort is to power physics-based simulation with generative capabilities, aiming for auto-generating robotic demonstration data. Genesis integrates two levels of system: a low-level underlying physics engine, and an upper-level generative agent that could autonomously create realistic physical world in simulation using the low level physics engine. We envision a pipeline where all the stages required for collecting robotic demonstration data in simulation could be automated: from intelligent task and skill proposal, to asset generation, scene creation, writing training supervisions such as reward functions, and automatically launching skill training via reinforcement learning or trajectory optimization. In addition to that, a long term goal of the Genesis initiative is to build a flexible data engine that also produces multi-modality data that can be potentially applicable to areas beyong robotics.

Our data engine is comprised of different modules that are integrated into a unified framework:

- **Robotic Demonstration Data Generation** This module integrates a complete pipeline from task proposal, to scene generation, reward generation, policy training, and demonstration generation. This module consists of systems developed in Gen2Sim [126] and RoboGen [291], a series of efforts as independent research projects to achieve this automation process, that we will describe in the next two chapters.

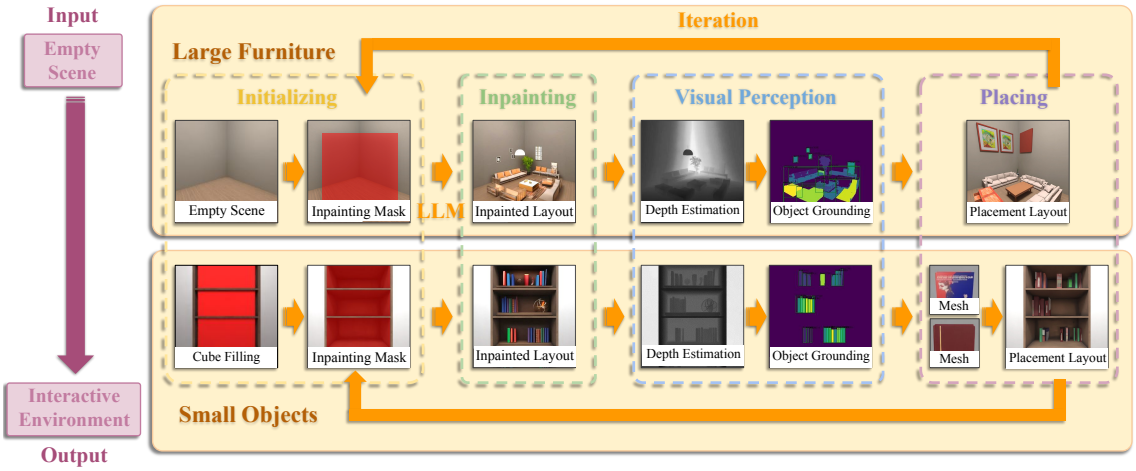


Figure 5.3: Architect pipeline.

Figure 5.4: Architect is a generative framework to create diverse, realistic, and complex Embodied AI scenes. Leveraging 2D diffusion models, we generate scenarios in an open-vocabulary manner. Here, we showcase two cases in detail: an apartment and a grocery store.

- **Scene Generation** We developped a top-down approach for generating complex and interactive 3D scenes that aims to match data distributions of the real world. Since pre-trained 2D image generative models better capture scene and object configuration than LLMs, we developped ARCHITECT, a generative framework that creates complex and realistic 3D embodied environments leveraging diffusion-based 2D image inpainting. We utilize foundation visual perception models to obtain each generated object from the image and leverage pre-trained depth estimation models to lift the generated 2D image to 3D space. In order to address challenges associate with absent camera parameters in the generated image, we control the diffusion model with hierarchical inpainting. Specifically, having access to ground-truth depth and camera parameters in simulation, we first render a photo-realistic image of only back-grounds in it. Then, we inpaint the foreground in this image, passing the geometric cues in the back-ground to the inpainting model, which informs the camera parameters. This process effectively controls the camera parameters and depth scale for the generated image, facilitating the back-projection from 2D image to 3D point clouds. Our pipeline is further extended to a hierarchical and iterative inpainting process to continuously generate placement of large furniture and small objects to enrich the scene. This iterative structure brings the flexibility for our method to generate or refine scenes from various starting points, such as text, floor plans, or pre-arranged environments. We integrated this module into genesis for generating high quality and diverse interactive environments. The whole pipeline is illustrated in Figure 5.3, and we show two example generated scenes in Figure 5.4. We refer readers to [289] for more details.
- **Character Motion & Facial Expression** We additionally would like to model non-robotic agents, such as the body motion of human characters and their facial expressions. To achieve this, we implemented two state-of-the-art generative modules for these modes: EMDM [336] for language-conditioned human character motion generation and FaceFormer [61] for facial motion generation.
- **Camera Motion** We adopt a simple camera motion parametrization scheme to generate camera motion, comprised of a few components: the parent frame

T_p of the camera frame, and distance d, azimuth angle of the camera α, its height with respect to the parent frame h, and a lookat frame T_l. For generating each trajectory segment of the camera motion, we sample two keyframe poses for T_p and T_l, as well as d, h, T_l. We then use either smoothstep function or Bezier curve smoothing to generate smoothed camera trajectory. This simple parametrization scheme allows us to generate natural camera motion segments such as panning and zooming. In out future, we will consider more complex combination of more camera motion primitives to allow composition into more artistic camera motions.

- **Articulated Object Generation** When generating interactive scenes for robotic agents to develop skills, it's important to also include objects with various forms of articulations. Previously, articulated asset population has been limited to PartNet Mobility [306], a dataset of human-defined articulated assets. However, creating scenes with open-world articulated asset will significantly enhance the diversity of the scenes and the way we can interact with them. 3D articulated objects generation has been a challenging problem in the literature, since it requires to capture both accurate surface geometries and semantically meaningful and spatially precise structures, parts, and joints. Existing methods heavily depend on training data from a limited set of handcrafted articulated object categories (e.g., cabinets and drawers), which restricts their ability to model a wide range of articulated objects in an open-vocabulary context. We propose Articulate-Anything, an automated framework that is able to convert any rigid 3D mesh into its articulated counterpart in an open-vocabulary manner. Given a 3D mesh, our framework utilizes advanced Vision-Language Models and visual prompting techniques to extract semantic information, allowing for both the segmentation of object parts and the construction of functional joints. Our experiments show that ARTICULATE ANYTHING can generate large-scale, high-quality 3D articulated objects, including tools, toys, mechanical devices, and vehicles, significantly expanding the coverage of existing 3D articulated object datasets. We illustrate the pipeline for this module in Figure 5.5.

Genesis is an ongoing project and will be continuously maintained and developped. The long-term goal of Genesis is to build a fundamental data source, where we can

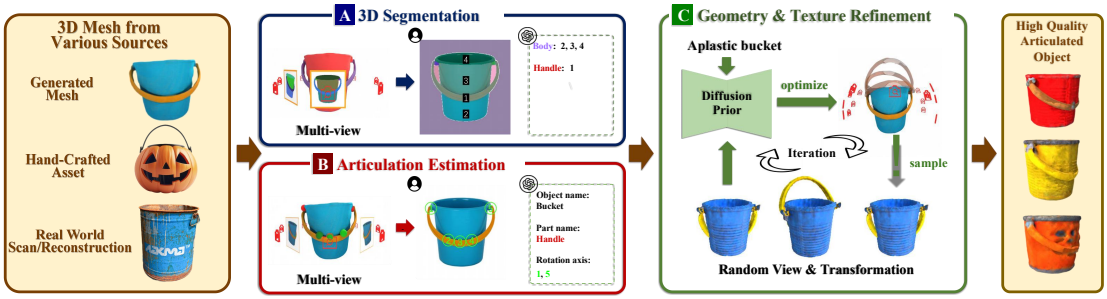


Figure 5.5: Articulate-Anything is able to convert any mesh into its articualted counterpart automatically.

extract various spatially-consistent, visually realistic, and physical accurate data. We believe this will make a significant impact on not only the robotics community, but also other relevant data-driven research fields.

Part II

Automating Robotic Demonstration Data Generation via Generative Simulation

Chapter 6

Gen2Sim: Scaling up Robot Learning in Simulation with Generative Models

6.1 Introduction

Scaling up training data has been a driving force behind the recent revolutions in language modeling [23], image understanding [216], speech recognition [218], image generation [229], to name a few. This begs the question: can we scale up robot data to enable a similar revolution in robotic skill learning? One way to scale robot data is in the real world, by having multiple robots explore [138] or by having humans provide kinesthetic demonstrations [21, 22, 253]. This is a promising direction; however, safety concerns and wear and tear of the robots hinder robot exploration in the real-world, and collecting kinesthetic demonstrations scales poorly as it is time-consuming and labor-intensive [21]. Another way to scale robot data is in simulation, by developing simulated environments, defining tasks and their reward functions, and training robot policies with reinforcement learning, augmenting visuals and physics parameters to facilitate transfer of policies to the real world [98]. Such sim2real paradigm has seen recent successes in robot locomotion [68, 118, 135, 137], object re-orientation [33, 203], and drone flight [128]. These examples, though very important and exciting, are still

fairly isolated.

A central bottleneck towards scaling up simulation environments and tasks is the laborious manual effort needed for developing the visuals and physics of assets, their spatial arrangement and configurations, the development of task definition and reward functions, or the collection of programmatic demonstrations. Tremendous resources have been invested in developing simulators for autonomous vehicles [53], warehouse robots, articulated objects [306], home environments [73, 236, 257], etc., many of which are proprietary and not open-sourced. Given these considerations, an important question naturally arises: How can we minimize manual effort in simulation development for diverse robotic skill learning?

In this chapter, we explore automating the development of simulation environments, manipulation tasks and rewards for robot skill learning, by building upon latest advances in large pre-trained generative models of images and language. Our system strives to automate all stages of robot learning: from generating 3D assets, textures, and physics parameters, to generating task descriptions and reward functions, leading to automated skill learning in diverse scenarios, as shown in Figure 7.1. This generative pipeline was first proposed in our position paper [302], described as a promising pathway towards generating diverse data for generalist robot learning. In this chapter, we present Gen2Sim, the first attempt and realization of such a generative paradigm. We automate 3D object asset generation by combining image diffusion models for 3D mesh and texture generation, and LLMs for querying physical parameters information. We showcase how LLMs and image generative models can diversify the appearances and behaviors of assets by producing plausible ranges of textures, sizes and physical parameters, achieving "intelligent" domain diversification. We automate task description, task decomposition and reward function generation by few-shot prompting of LLMs to generate language descriptions for semantically meaningful tasks, concerning affordances of existing and generated 3D assets, articulated or not, alongside their reward functions. Gen2Sim is able to generate numerous object assets and task variations without any human involvement beyond few LLM prompt designs. We successfully train RL policies using our auto-generated tasks and reward functions. We also demonstrate the usefulness of our simulation-trained policies, by constructing digital-twin environments from given real scenes, allowing a robot to practice skills in the twin simulator and deploying it back

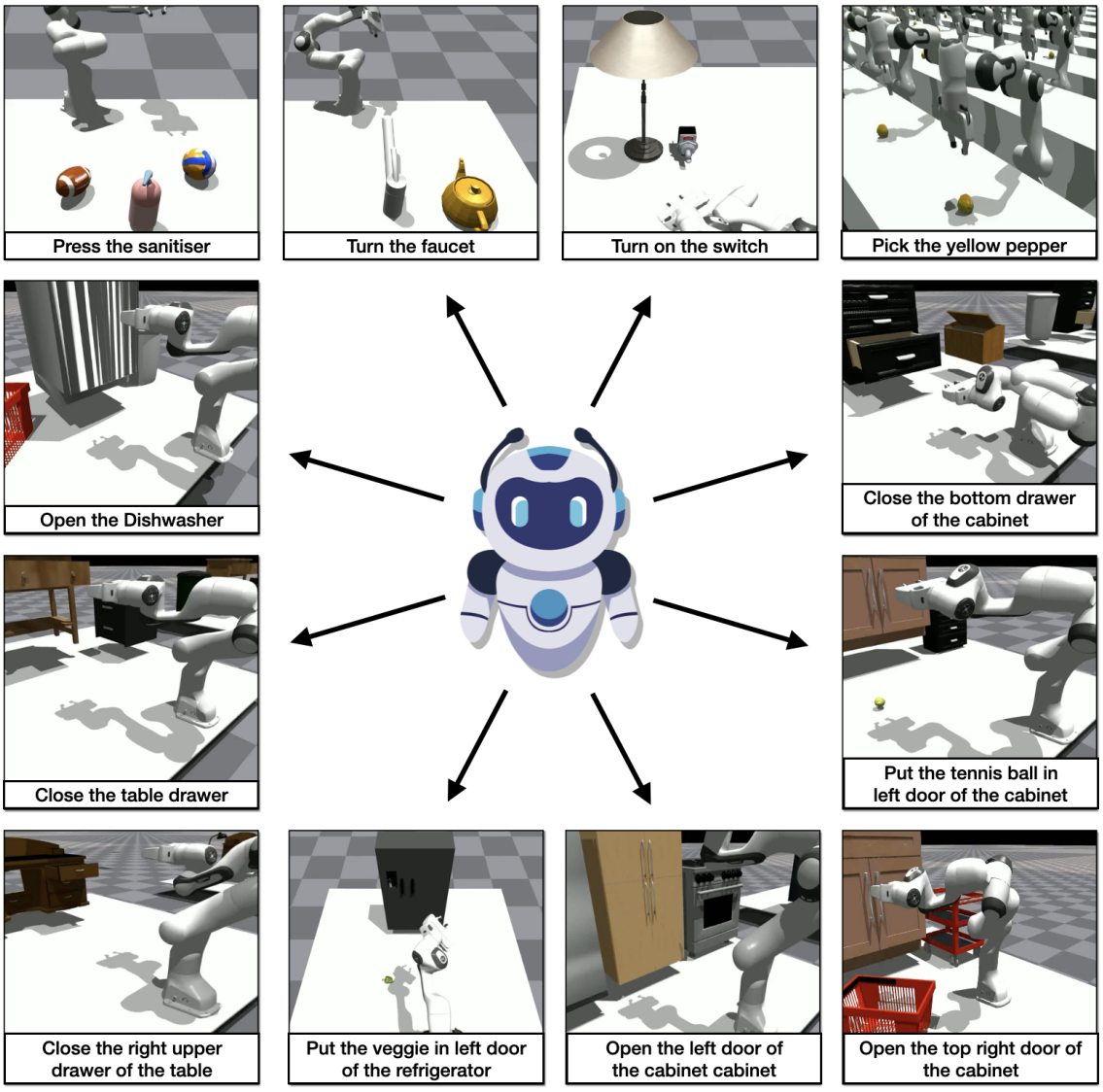


Figure 6.1: **Gen2Sim** is an automated generative pipeline of assets, tasks, task decompositions, and rewards functions for autonomous robotic skill reinforcement learning in simulation. Here we show 12 generated tasks, concerning affordances of diverse types of object assets and their combinations.

to the real world to execute the task.

In summary, we make the following contributions:

- We show how pre-trained generative models of images and language can help automate 3D asset generation and diversification, task description generation, task decomposition and reward function generation that supports reinforcement

learning of long horizon tasks in simulation with minimal human involvement.

- We deploy our method to generate hundreds of assets, and hundreds of manipulation tasks, their decompositions and their reward functions, for both human-developed and automatically generated object assets.

6.2 Related Work

Large Language Models for task and motion planning in robotics Large language models (LLMs) map instructions to language subgoals [104, 105, 307, 341] or action programs [148] with appropriate plan-like or program-like prompts. LLMs trained from Internet-scale text have shown impressive zero-shot reasoning capabilities for a variety of downstream language tasks [23] when prompted appropriately, without any weight fine-tuning [24, 27, 161, 293]. LLMs were used to generate task curricula and predict skills to execute in Minecraft worlds [151, 283, 338] Following the seminal work of Code as Policies, many works map language to programs over given skills [84] or hand-designed motion planners [107]. Our work instead maps task descriptions into task decompositions and reward functions, to guide reinforcement learning in simulation, to discover behaviours that achieve the generated tasks. Work of [324] also uses language for predicting reward functions for robot locomotion, but does not consider task generation and decomposition or interaction with objects. Our work is the first to use LLMs for task decomposition and reward generation, as well as asset generation.

Automating 3D asset creation with generative models The traditional process of creating 3D assets typically involves multiple labor-intensive stages, including geometry modeling, shape baking, UV mapping, material creation, texturing and physics parameter estimation, where different software tools and the expertise of skilled artists are often required. It is thus desirable to automate 3D asset generation to automatically generate high-quality assets that support realistic rendering under arbitrary views and have plausible physical behaviours during force application and contacts. The lack of available 3D data and the abundance of 2D image data have stimulated interest in learning 3D models from 2D image generators [26, 196]. The availability of strong 2D image generative models based on diffusion led to high-quality

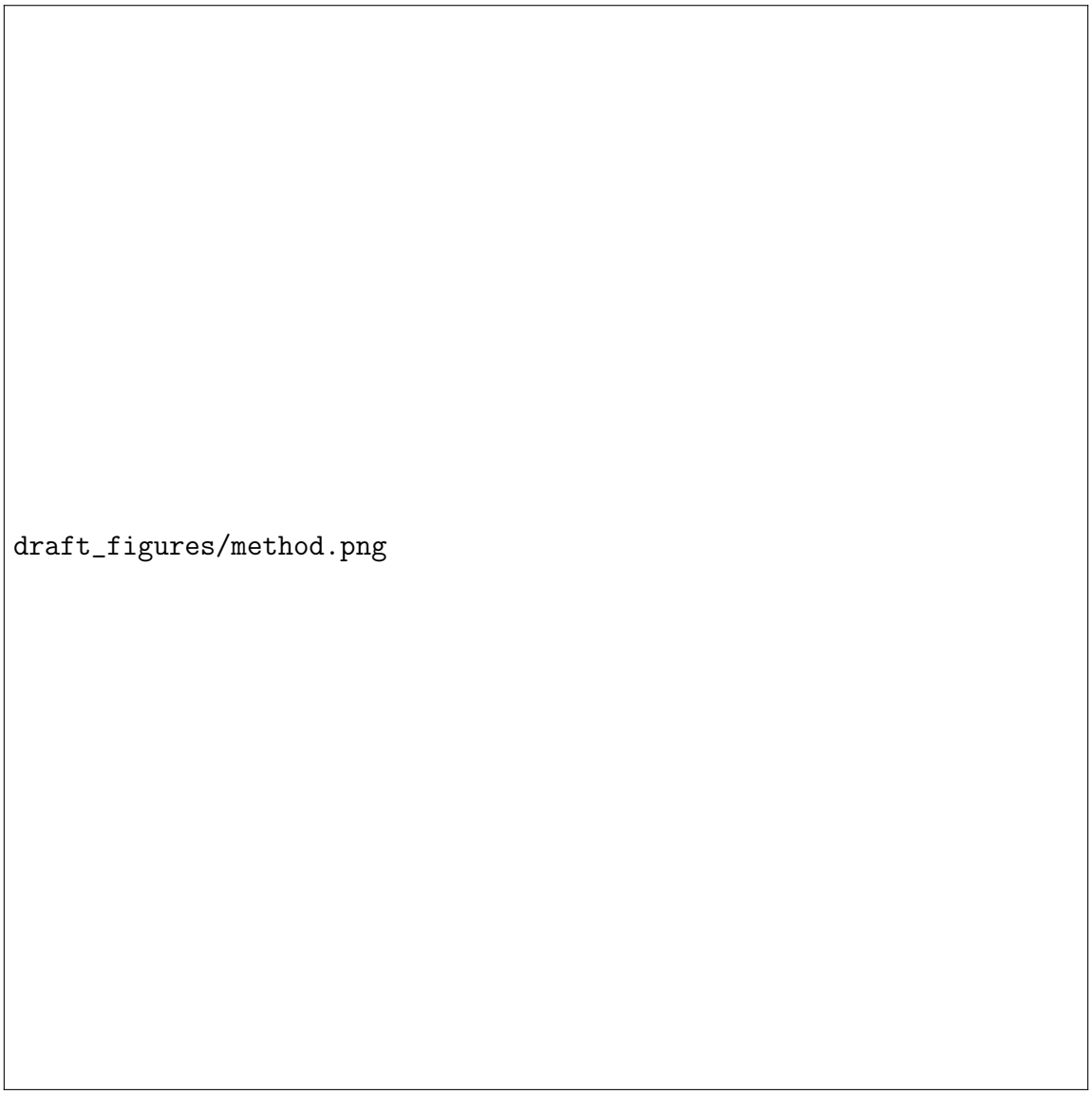


Figure 6.2: **The Gen2Sim components.** Gen2Sim generates 3d assets by lifting object-centric 2D images to 3D. It then uses both generated assets and assets obtained from other publicly available datasets to populate scene environments. Afterwards, it queries LLMs to generate meaningful task descriptions for the assembled scenes, as well as decompose the generated task descriptions to sub-tasks and their reward functions.

3D models from text descriptions [31, 152, 214] or single 2D images using the diffusion model as a 2D prior [181, 246, 267]. In this work, instead of a text-conditioned model,

we use a view and relative pose conditioned image generative model, which we found to provide better prior for score distillation. Some methods attempt to use videos of assets and differentiable simulations to estimate their physics parameters and/or adapt the simulation environment, in an attempt to close the simulation to reality gap [95, 96, 285]. Our effort is complementary to these works.

Procedural demonstration generation using symbolic planners Many recent works procedurally generate scenes and demonstration trajectories using planners that have access to privileged information to solve the task, and distill the demonstration solutions into learning-based policies that operate directly from pixel or point-cloud input [46, 66, 179]. Task and motion planners [121, 171, 185, 274] use predefined symbolic rules and known dynamics models, and infer discrete task plans given instruction with lookahead logic search [74, 121, 121, 171, 185, 274]. These methods predominantly rely on manually-specified symbolic transition rules, planning domains, and grounding, which limits their applicability. Indeed, works of [46, 179] demonstrate their results on relatively simple multi-object box stacking tasks. Scene procedural generation in the aforementioned works [46, 179, 189] entails randomizing locations and number of given 3D models under weak supervision from a human that defines the task and the possible location candidates. In contrast, we unleash the common sense knowledge and reasoning capabilities provided by LLMs and use them to suggest task descriptions, task decompositions, and reward functions. We then use reinforcement learning to discover solution trajectories instead of TAMP-based search.

Simulation environments for robotic skill learning In recent years, improving simulators for robot manipulation has attracted increasingly more attention. Many robotic manipulation environments and benchmarks [20, 132, 306] are built on top of either PyBullet [44] or MuJoCo [273] as their underlying physics engines, which mainly support rigid-body manipulation [76, 78, 278, 301, 303]. Recently, environments supporting soft-body manipulation ([72, 155, 175, 287, 305, 306]) provide capabilities for simulating deformable robots, objects and fluids. Our automated asset and task generation are not tied to any specific simulation platforms and can be used with any of them.

6.3 Gen2Sim

Gen2Sim generates 3D assets from object-centric images using image diffusion models and predicts physical parameters for them using LLMs (Section 6.3.1). It then prompts LLMs to generate language task descriptions and corresponding reward functions for each generated or human-developed asset, suitable to their affordances (Section 6.3.2). Finally, we train RL policies in the generated environments using the generated reward functions. We additionally show the applicability of the simulation-trained policy by constructing digital twin environment in simulation, and deploy the trained trajectory in the real world (Section 6.3.3). See Figure 6.2 for our method overview.

6.3.1 3D Asset Generation

Gen2Sim automates 3D asset generation by mapping 2D images of objects to textured 3D meshes with plausible physics parameters. The images can be 1) real images taken in the robot's environment, 2) real images provided by Google search under relevant category names, e.g., *"avocado"*, or 3) images generated by pre-trained text-conditioned diffusion models, such as stable diffusion [230], prompted appropriately to generate uncluttered images of the relevant objects, e.g., *"an image of an individual avocado"*. We query GPT-4 [202] for a list of object categories relevant for manipulation tasks to search online for or to generate, instead of manually designing it. Please, visit our project site for a detailed list of the objects we generated. Given a real or generated 2D image of an object, we lift it to a 3D model by minimizing re-reprojection error and maximizing likelihood of its image renderings using a diffusion model [31, 214]. We provide background on image diffusion models below, before we describe our 3D model fitting approach.

Image diffusion models

A diffusion model learns to model a probability distribution $p(x)$ by inverting a process that gradually adds noise to the image x. The diffusion process is associated with a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$, which defines how much noise is added at each time step. The noisy version of sample x at time t can then be written

$x_t = \sqrt{\bar{\alpha}_t}x + \sqrt{1-\bar{\alpha}_t}\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, is a sample from a Gaussian distribution (with the same dimensionality as x), $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. One then learns a denoising neural network $\hat{\epsilon} = \epsilon_\phi(x_t; t)$ that takes as input the noisy image x_t and the noise level t and tries to predict the noise component ϵ. Diffusion models can be easily extended to draw samples from a distribution $p(x|\mathbf{c})$ conditioned on a prompt $\mathbf{c}$, where $\mathbf{c}$ can be a text description, a camera pose, and image semantic map, *etc* [144, 229, 332]. Conditioning on the prompt can be done by adding $\mathbf{c}$ as an additional input of the network ϵ_ϕ. For 3D lifting, we build on Zero-1-to-3 [162], a diffusion model for novel object view synthesis that conditions on an image view of an object and a relative camera rotation around the object to generate plausible images for the target object viewpoint, $\mathbf{c} = [I_1, \pi]$. It is trained on a large collection $\mathcal{D}' = \{(x^i, \mathbf{c}^i)\}_{i=1}^{N}$ of images paired with views and relative camera orientations as conditioning prompt by minimizing the loss:

$$\mathcal{L}_{\text{diff}}(\phi; \mathcal{D}') = \frac{1}{|\mathcal{D}'|} \sum_{x^i, \mathbf{c}^i \in \mathcal{D}'} ||\epsilon_\phi(\sqrt{\bar{\alpha}_t}x^i + \sqrt{1-\bar{\alpha}_t}\epsilon, \mathbf{c}^i, t) - \epsilon||^2.$$

Image-to-3D Mesh using Score Distillation Sampling

Given an image and relative camera pose 2D diffusion model $p(I|[I_0, \pi])$, we extract from it a 3D rendition of the input image I_0, represented by a differential 3D representation using Score Distillation Sampling (SDS) [214, 284]. We do so by randomly sampling a camera pose π, rendering a corresponding view I_π, assessing the likelihood of the view based on a diffusion model $p(I_\pi|[I_0, \pi])$, and updating the differentiable 3D representation to increase the likelihood of the generated view based on the model. Specifically, the diffusion model is frozen and the 3D model is updated as:

$$\nabla(\theta)\mathcal{L}_{SDS}(\theta; \pi, \mathbf{c}, t) = \\ \mathbb{E}_{t,\epsilon}[w(t)(\epsilon_\phi(a_t I + \sigma_t \epsilon; t, \mathbf{c}) - \epsilon) \cdot \nabla_\theta I],$$

where $I = R(\theta, \pi)$ is the image rendered from a given viewpoint π. The loss we use to backpropagate to the 3D model parameters θ includes an image re-projection loss for the camera viewpoint of the input image, and score distillation for the other views, using a pre-trained view and pose conditioned image diffusion model of [162] to measure 2D image likelihood. We use a two-stage fitting, wherein the first stage

an instantNGP NeRF representation [**?**] is used, similar to RealFusion [181], and in the second stage a mesh-based representation is initialized from the NeRF and finetuned differentiably, similar to Fantasia3D [31]. More information of our score distillation sampling can be found in our website.

Texture generation

We augment the textures of our generated assets using the method of TEXTure [225] which iteratively edits a mesh's texture by rendering the mesh from different viewpoints and updating the rendered 2D images. While domain randomization [271] randomly re-textures simulated assets, TEXTure produces diverse yet plausible texture augmentations.

Generating plausible physical properties

The visual and collision parameters of an asset are generated from the Image-to-Mesh pipeline discussed above. To define 3D sizes and physics parameters for the generated 3D meshes, we query GPT-4 regarding the range of plausible width, height, and depth for each object, and the range of its mass given its category. We then scale the generated 3D mesh based on the generated size range. We feed the mass and 3D mesh information to MeshLab [41] to get the inertia matrix for the asset. Our prompts for querying GPT for mass and 3D object size can be found on our website. We wrap the generated mesh information, its semantic name, as well as the physical parameters into URDF files to be loaded into our simulator.

6.3.2 Task Generation, Temporal Decomposition and Reward Function Prediction

Given either generated assets or assets obtained from publically available datasets, we prompt LLMs to generate meaningful manipulation tasks considering their affordances, to decompose these tasks into subtasks when possible, and to generate reward functions for each subtask. We train reinforcement learning policies for each (sub)task using the generated reward functions, and then chain them together to solve long horizon tasks. Our LLM prompts contain the following sections:

1. Asset descriptions. We use combinations of assets we generate using the method of Section 6.3.1, as well as articulated assets from PartNet Mobility [306] and GAPartNet dataset [75]. We populate our simulation environment with randomly sampled assets. Then, we extract information from the URDF files including link names, joint types and limits using automated scripts. For example, an asset `microwave` has parts [`door`, `handle`, and `body`], and joint [`door-joint`] of type `revolute` with a joint position range $[0, 1]$. We then describe the extracted configurations of the assets to the LLM, as shown below:

```
The environment contains the following assets:
1.  asset_name: "microwave"
    part_cofiguration:
        Part 1: "body"
        Part 2: "door"
            - link_name: "link_0"
            - joint_name: "joint_0"
            - joint_type: "revolute"
            - limit: [0, 1]
        Part 3: "handle"
            - link_name: "handle_0"
            - joint_name: "handlejoint_0"
            - joint_type: "fixed"
2.  asset_name: "cup"
    part_cofiguration:
        Part 1: cup
            - link_name: "base"
            - joint_name: "base_joint"
            - joint_type: "fixed"
```

2. Instructions. These include function APIs that can be used by the LLM to query the pose of the robot end-effector, as well as different assets in the given environment:

```
Available APIs from the simulator are:
# returns the pose of the link
get_pose_by_link_name(asset_name, link_name)
# returns the pose of the robot gripper
get_robot_gripper_pose(asset_name, link_name)
# returns the state of the joint
get_state_by_joint_name(asset_name, joint_name)
# returns the limit of the joint
```

```
get_limits_by_joint_name(asset_name, joint_name)

Note:
1. Only use the available APIs from the simulator.
2. Generate the reward function code snippets in Python.
```

3. Examples of task descriptions and decompositions. These are question-answer pairs that demonstrate task descriptions and their temporal decompositions.

```
List meaningful manipulation tasks that can be performed
in this environment. Give subtask decomposition and the
order of execution to solve the task. Also, provide the
reward function for each subtask.

The following tasks can be performed in this environment:
1. Open the Microwave Door
2. Close the Microwave Door
3. Pick Cup
4. Place Cup
5. Put the Cup in the Microwave
   This task needs to be decomposed into sub-tasks:
    - Open the Microwave
    - Pick Cup
    - Place the Cup in the Microwave
```

4. Examples of reward functions. These are task to reward function pairs that present demonstrations of how tasks can be translated to reward functions, as shown below:

```
Task: OpenMicrowaveDoor
Task Description: open the door of the microwave
'''
def compute_reward(env):
    # reward function
    door_handle_pose = env.get_pose_by_link_name("microwave", "
   handle_0")
    gripper_pose = env.get_robot_gripper_pose()
    distance_gripper_to_handle = torch.norm(door_handle_pose -
   gripper_pose, dim=-1)
    door_state = env.get_state_by_joint_name("microwave", "joint_0")
    cost = distance_gripper_to_handle - door_state
```

```
    reward = - cost

    # success condition
    target_door_state = env.get_limits_by_joint_name("microwave", "joint_0")["upper"]
    success = torch.abs(door_state - target_door_state) < 0.1

    return reward, success
```

For the example above, the reward function is comprised of 1) distance between the end-effector and the target part, and 2) distance between the current and the target pose of an articulated asset, link, or joint.

We provide our prompts on our website. We show in Section 8.5 that our method can generalize across assets, suggest diverse and plausible tasks, decomposition and reward functions automatically, using a single in-context example in the prompt, without any additional human involvement.

6.3.3 Sequential Reinforcement Learning for Long Horizon Tasks

We train policies using Proximal Policy Optimization (PPO) [238] maximizing the generated reward functions for each subtask. We train RL for each generated subtask in temporal order. Once policy training for a subtask converges, we proceed to the next subtask, by sampling the initial state of the end-effector and the environment close to the terminal states of the previous subtask. This ensures policies can be temporally chained upon training. Our policies are trained per environment using privileged information of the simulation state to accelerate exploration. Such learned policies can be used as demonstration data and distilled into vision-language transformer policies, similar to [21, 40, 111]; we leave this for future work.

6.4 Experiments

Our experiments aim to answer the following questions:

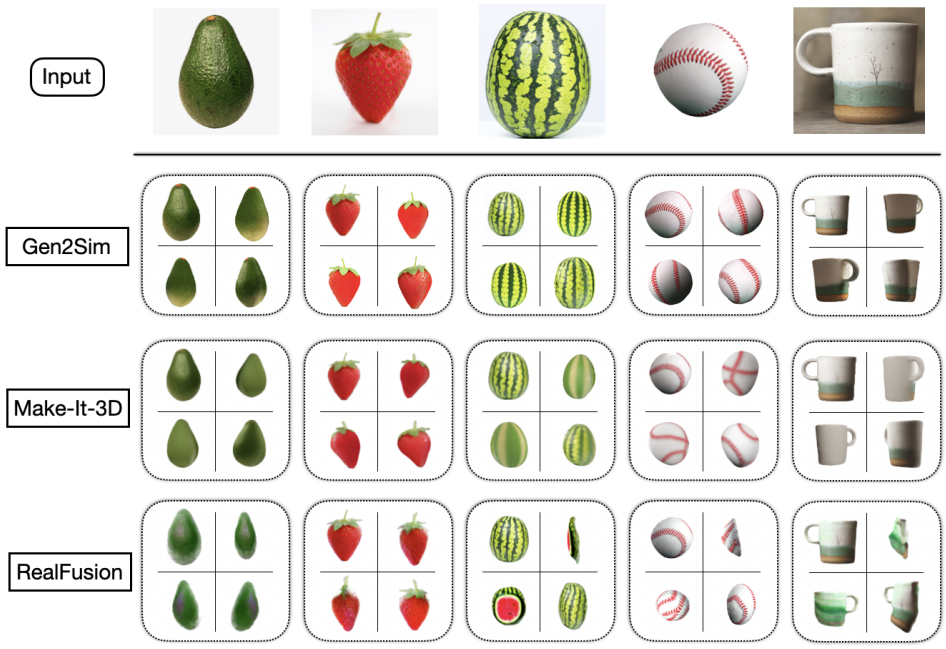


Figure 6.3: **3D asset generation** from Gen2Sim, RealFusion [181] and Make-It-3D [267]. Gen2Sim uses a view and camera pose conditioned image generative model during score distillation, which helps generate more accurate 3D geometry in comparison to the baselines.

1. Can Gen2Sim generate plausible geometry, appearance, and physics for diverse types of objects and parts, without human expertise and with minimal human involvement?

2. Can Gen2Sim generate task language goals and reward functions for novel object categories, novel assets with different part configurations, and a combination of multiple assets in an environment?

3. Can the generated environments and reward function lead to successful learning of RL policies?

6.4.1 Asset Generation

We compare our image-to-3D lifting with two baselines:

1. *RealFusion* [181], which uses textual inversion of [70] to learn a word embedding for the depicted object concept in an image, and uses text-conditioned diffusion with this text embedding during score distillation.

	Mass (gram)	Length (cm)	Width	Height
Papaya	500-1000	15-20	10-15	10-15
Cucumber	200-300	15-20	5-7	5-7
Watermelon	5000-7000	30-40	20-30	20-30
Raspberry	3-5	2-3	2-3	2-3
Coconut	600-800	10-15	8-12	8-12
Corn	50-100	10-15	8-12	8-12
Pumpkin	2000-5000	20-40	20-40	20-40
Avocado	150-250	10-12	6-8	4-5

Table 6.1: **Size and physics parameter generated by LLMs** for a number of generated assets.

2. *Make-It-3D* [267], which uses the same NeRF and textured mesh two-stage fitting as Gen2Sim, but does not use a view and pose conditioned generative model, rather a text-based image diffusion model, similar to [214].

We show comparisons in Figure 6.3, with images rendered from 4 different views. Our model generates more plausible 3D model as our image diffusion prior comes from an image and pose-conditioned model in comparison to approaches like Fantasia3D or RealFusion which uses text conditioning. We show generated values for 3D sizes and mass for a number of example objects in Table 6.1. We see that the common sense knowledge encoded in LLMs can produce reasonable physical parameters.

6.4.2 Automated Skill Learning

Gen2Sim generates diverse task descriptions, task decompositions and reward functions automatically for hundreds of assets, with different category labels and number of joints, **given only a single in-context prompt example** regarding the task decomposition and reward function of the task "putting a cup in a Microwave" . Then, the model can generalize to different scenes, asset articulated structures and task temporal lengths. We show some examples of such generated task descriptions in Figure 7.1 and more on our website. We show examples of task decompositions in Figure 6.2. We provide our prompts in our project website, alongside examples of the LLM's responses.

We learn policies that optimize LLM generated rewards with PPO, an off-the-shelf

model-free RL algorithm [238]. We make use of GPU-parallel data sampling in IsaacGym [176] for reinforcement learning. Our robotic setup uses a Franka Panda arm with a mobile base. It is equipped with a parallel-jaw gripper. Our state representation for PPO includes the robot's joint position $q \in \mathbb{R}^{11}$, velocity $\dot{q} \in \mathbb{R}^{11}$ (7-DoF arm, x and y for the mobile base and 2 extra DoFs from the gripper jaws), orientation of the gripper $r \in SO(3)$, and poses and joint configurations of the assets present in the scene. We use position control and at each timestep t our policy produces target gripper pose and configurations which is converted to target robot configurations through inverse kinematics. A low-level PID torque controller provided by IsaacGym is used to produce low-level joint torque commands. We can successfully learn useful manipulation policies, and the polices are able to solve the tasks upon convergence. We show videos of such policies on our website.

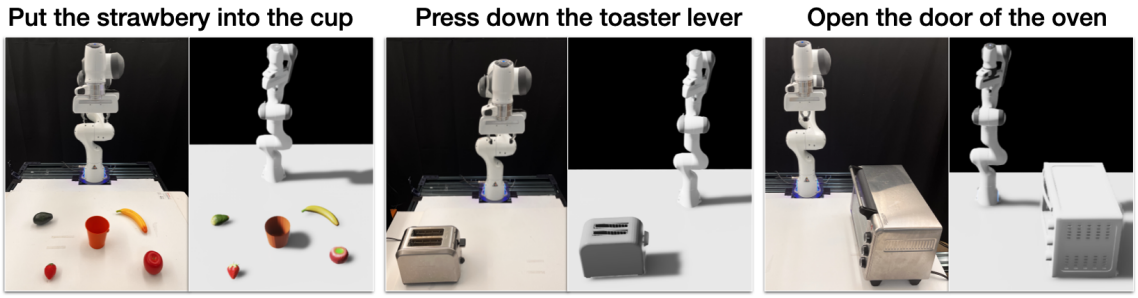


Figure 6.4: **Twin environments** constructed and generated tasks for sim-to-real transfer. Left: real-world. Right: simulated.

6.4.3 Twin environment construction and sim-to-real world transfer

In order to validate the usefulness of the policies trained in simulation, we construct a twin simulated environment of our lab's real-robot setup (Figure 6.4). We detect, segment, and estimate the poses of the objects in the scene. For non-articulated assets, we use our model to lift the detected object image to corresponding 3D models; for articulated objects, we select the most similar asset from the [306], and populated the simulated environment. We train RL policies in simulation and transfer the joint space trajectory back to our real-world setup. Our method allows successful

execution of the generated tasks. For more videos of the trained policies and their task executions in simulation, as well as the sim2real transfer, please refer to our website.

6.4.4 Limitations

Gen2Sim has currently the following two important points to address towards materializing into a platform for large-scale robot skill learning that are deployable in real-world:

1. Sim2real transfer of closed-loop policies: Our current real-world experiments transfer open loop trajectories optimized in the constructed twin environment. For closed-loop policies to transfer to the real world and consume realistic sensory input, we would need to generate large-scale augmentations for both visual appearances and dynamics for each task and sub-task, and then distil the state-based RL policies to a foundational vision-language policy network. This is a direct avenue for our future work.

2. Beyond rigid asset generation: The assets we can currently generate are rigid or mostly rigid objects, which do not deform significantly under external forces. For articulated assets, we are using existing manually designed and labelled datasets ([75, 306]). To generate articulated objects, deformable objects and liquids, accurate fine-grained video perception is required in combination with generative priors to model the temporal dynamics of their geometry and appearance. This is an exciting and challenging direction for future work.

6.5 Conclusion

We have presented Gen2Sim, a method for automating the development of simulation environments, tasks and reward functions with pre-trained generative models of vision and language. We presented methods that create and augment geometry, textures and physics of object assets from single images, parse URDF files of assets, generate task descriptions, decompositions and reward python functions, and train reinforcement learning policies to solve the generated long horizon tasks. Addressing the limitations including generating diverse assets with more complex physical properties, and

transfering trained policies to real world are direct avenues for our future work. We believe generative models of images and language will play an important role in automating and supersizing robot training data in simulation, and in crossing the sim2real gap, necessary for delivering robot generalists in the real world. Gen2Sim takes one first step in that direction.

Chapter 7

RoboGen: A More Comprehensive Robotic Data Generation System for Diverse and Automated Robot Learning

7.1 Introduction

Simulated environments have become a crucial driving force for teaching robots various complex skills, spanning complex manipulation and locomotion settings [33, 86, 294, 313, 342]. Compared to exploration and data collection in the real-world, simulated environments provide access to privileged low-level states and unlimited explorations, and support massively parallel computation for significantly faster data collection without considerable investment in robotic hardware. However, robot learning in simulations also presents its own limitations: while exploration and practicing in simulated environments are cost-effective, constructing these environments requires tremendous human effort, demanding tedious steps including designing tasks, producing relevant and semantically meaningful assets, generating plausible scene layouts and configurations, and crafting training supervisions such as reward or loss functions [80, 113, 139, 258]. The onerous task of creating these components and constructing

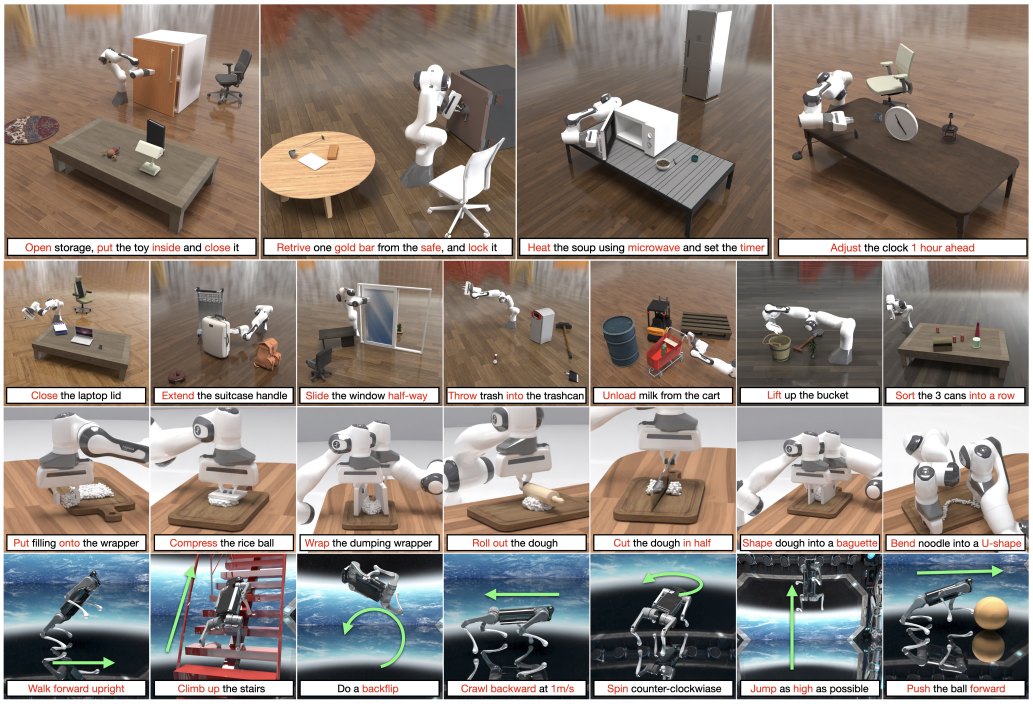


Figure 7.1: 25 example tasks generated and corresponding skills learned by RoboGen. Readers are encouraged to visit our project website for the diverse set of tasks and skills RoboGen can produce.

individualized simulation settings for each one of the countless tasks encountered in our daily life significantly hinders the scalability of robotic skill learning even in simulated worlds.

In light of this, we propose *Generative Simulation* [302], a new paradigm aiming for scaling up simulated robot learning with the latest advancement in generative models. Generative simulation advocates for *autonomously* generating information for all the stages needed for diverse robotic skill learning in simulation: from high-level task and skill proposals to task-dependent scene descriptions, asset selections and generations, policy learning choices, and training supervisions. These information is then used for massive skill training, enabling robots to acquire proposed skills. In this chapter, we present *RoboGen*, a robotic agent that continuously generates new skills via a self-guided *propose-generate-learn* cycle: it firstly self-proposes skills to learn, and then generates required assets and constructs the scene in simulation conditioned on the proposed task. Afterwards, it labels the tasks with natural language

descriptions, decomposes the task into sub-tasks, selects the optimal learning approach (reinforcement learning, motion planning, or trajectory optimization), designs proper training supervisions (e.g. reward functions), and lastly proceeds to policy learning to solve the proposed task. One distinct advantage of our proposed paradigm lies in the careful choice of what modes of knowledge to extract from contemporary foundation models. These models have demonstrated impressive capabilities across various modalities [55, 122, 201, 229, 275], However, due to the absence of training data pertaining to *dynamics*, *actuations*, and *physical interactions*, these models are yet to develop essential understandings for robots to execute physical actions and interact with the surrounding environments (e.g., producing precise joint torques needed for walking or rolling a dough at hand). In contrast to recent efforts that employ foundation models such as Large Language Models (LLMs) for directly yielding policies or low-level actions [107, 149, 288], our method only extracts information that falls neatly within the capabilities and modalities of these models - object semantics, object affordances, common-sense knowledge regarding what tasks are valuable to learn, etc. These knowledge are used to construct environmental playgrounds, and then augmented with additional help from physics-grounded simulations, for robots to develop understandings of physical interactions and acquire diverse skills.

Our experiments show that RoboGen can deliver a continuous stream of diversified skill demonstrations, spanning tasks including rigid and articulated object manipulation, deformable object manipulation, as well as legged locomotion (see Figure 7.1). The diversity of tasks and skills generated by RoboGen surpasses previous human-crafted robotic datasets, with minimal human involvement beyond several prompt designs and in-context examples. Our work attempts to transfer the extensive and versatile knowledge embedded in large-scale models to the field of robotics, making a step towards automated large-scale robotic skill training and demonstration collection for building generalizable robotic systems.

7.2 Related Work

Robotic skill learning in simulations Various physics-based simulation platforms have been developed in the past to accelerate robotics research [158]. These include rigid-body simulators [17, 44, 273, 306], deformable object simulators [94, 155, 175,

313], and environments supporting multi-material and their couplings with robots [73, 80, 305]. Such simulation platforms have been heavily employed in the robotics community for learning diverse skills, including deformable object manipulation [156, 290, 294], object cutting [94, 313], fluid manipulation [241, 305], as well as highly dynamic and complex skills such as in-hand re-orientation [7, 33], object tossing [328], acrobatic flight [127, 168, 256], and legged locomotion [35, 219, 342].

Scaling up simulation environments Apart from building physics engines and simulators, a large body of prior work targeted at building large-scale simulation benchmarks, providing platforms for scalable skill learning and standardized benchmarking [80, 113, 139, 155, 258, 305, 322]. Notably, most of these prior simulation datasets are manually built with human labeling. Another line of works attempts to scale up tasks and environments using procedural generation, and generate demonstrations with Task and Motion Planning (TAMP) [46, 120, 179, 189]. These methods primarily build on top of manually-defined rules and planning domains, limiting the diversity of the generated environments and skills to relatively simple pick-and-place tasks [46, 179]. Contrary to these works, we leverage the common sense knowledge embedded in foundation models to generate meaningful tasks, relevant scenes, and skill training supervisions, leading to more diverse and plausible skills.

Foundation and generative models for robotics Following the advancement in foundation and generative models in domains of imagery, language and other modalities, [55, 77, 160, 180, 201, 214, 275], a line of works explores using these models for robotics research via approaches such as code generation [149, 298], data augmentation [323], visual imagination for skill execution [56], sub-task planning [3, 106, 153], concept generalization of learned skills [22], outputting low-level control actions [288], and goal specification [120, 124]. Related to ours are recent methods using LLMs for reward generation [173, 324], and sub-task and trajectory generation [84]. Concurrent work [286] explored LLM-based task generation, but are limited to table-top rigid object manipulation tasks with limited assets. The task demonstrations are generated by using LLMs to directly write the code script for manipulating the objects; in contrast, we use LLMs to generate the rewards, invoke appropriate algorithms (motion planning, RL, etc) to learn the skill and generate the demonstrations, which is more general. Katara et al. [126] also used a LLM to generate table-top rigid and articulated object manipulation tasks and rewards. Ours

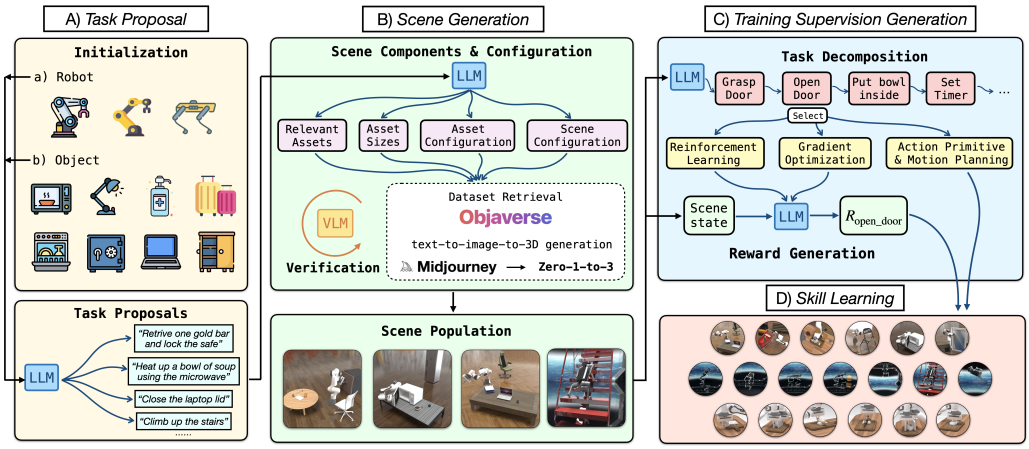


Figure 7.2: RoboGen consists of the following stages: A) task proposal, B) scene generation, C) training supervision generation, and D) skill learning with generated information.

differ as we additionally perform scene generation and automatic algorithm selection. We also demonstrate our pipeline with more diverse tasks, including more complex and long-horizon articulated object manipulation tasks, as well as locomotion and soft-body manipulation tasks.

7.3 RoboGen

RoboGen is an automated pipeline that utilizes the embedded common sense and generative capabilities of the latest foundation models [200, 268] for automatic task, scene, and training supervision generation, leading to diverse robotic skill learning at scale. We consider tasks including rigid (articulated) object manipulation, soft body manipulation, and legged locomotion. We illustrate the whole pipeline in Figure 7.2, composed of several integral stages: *Task Proposal*, *Scene Generation*, *Training Supervision Generation*, and *Skill Learning*. We detail each of them in the following.

7.3.1 Task Proposal

RoboGen starts with proposing meaningful and diverse tasks for robots to learn. We initialize the system with a specific robot type and an object randomly sampled from a pre-defined pool. The provided robot and sampled object information are then used

as input to an LLM to generate task proposal. This initialization step serves as a *seeding* stage, providing a basis upon which the LLM can condition and subsequently reason and extrapolate to generate a variety of tasks, taking into account both robot capability and object affordances. Apart from object-based initialization, another choice is to employ example-based initialization, where we initialize the query with a provided robot and several example tasks sampled from a list of 11 pre-defined tasks. For tasks involving legged robots and soft-body manipulation, we prompt the LLM with only example-based seeding.

We use GPT-4 [201] as our LLM backend to query in the current pipeline, which can be upgraded once better models are available. In the following, we explain details of RoboGen in the context of a robotic arm (e.g., Franka) and tasks generated pertain to object manipulation, using object-based initialization. In this case, the objects used for initialization are sampled from a predefined list, including common articulated and non-articulated objects in household scenarios such as oven, microwave, dispenser, laptop, dishwasher, etc., extracted from PartNetMobility [306] and RLBench [113]. The common sense and reasoning capability embedded in LLMs like GPT-4 allow them to produce meaningful tasks considering the object affordances, functionalities, and how they can be interacted with. We instantiate a prompt for task proposal containing the following information: 1) the category of the sampled object, 2) its articulation tree derived from its URDF file, and 3) semantic annotations of the links in the object's articulation, e.g., which link corresponds to the door in a sampled microwave. These information are provided by the PartNetMobility dataset. Additionally, we include one example input-output pair in the prompt. We feed the prompt to GPT-4 to obtain a number of semantically meaningful tasks that can be performed with the sampled object, where each task consists of 1) task name, 2) a natural language description of the task, 3) *additional* objects needed for performing the proposed task and 4) joints and links of the sampled articulated object relevant to the task.

As a concrete example, if a sampled articulated object is a `microwave`, where `joint_0` is a revolute joint connecting its door, and `joint_1` is another revolute joint controlling the timer knob, GPT-4 could return a task named "heat up a bowl of soup", with a task description of "The robot arm places a bowl of soup inside the microwave, closes the door and sets the microwave timer for an appropriate heating

duration", additional objects that are necessary for the generated task such as "A bowl of soup", and task-relevant joints and links including `joint_0` (for opening the microwave door), `joint_1` (for setting the timer), `link_0` (the door), and `link_1` (the timer knob). Note that for cases where we sample non-articulated objects or use example-based initialization, the sampled objects and examples are provided only as a hint for task proposal, and the generated tasks will not be tied to them. By repeatedly querying with different sampled objects and examples, we can generate a diverse range of manipulation and locomotion tasks, concerning the relevant object affordances when needed.

7.3.2 Scene Generation

Once a task proposal is obtained, RoboGen then generates a corresponding scene for solving the task by populating the environment with a number of relevant and necessary objects (*assets*). As shown in Figure 7.2 (B), generating a corresponding scene requires obtaining information for 4 different components: a) **relevant assets** to be used, b) **asset sizes**, c) **initial asset configurations** and d) **initial scene configuration**. We explain details in the following.

Relevant assets In the previous stage of task proposal, we obtained a list of relevant assets that are necessary for performing the proposed task. To further increase the complexity and diversity of the generated scenes while resembling object distributions of real-world scenarios, we query GPT-4 to return a number of additional queries (object names and their descriptions) that are semantically relevant to the task. For example (Figure 7.1), for the task "Open storage, put the toy inside and close it", the generated scene involves additionally a living room mat, a table-top lamp, a book, and an office chair. These queries (names) of the assets needed for the scene are used to search in existing object mesh databases. Specifically, we use Objaverse [51], a large-scale dataset containing over 800k object assets (3d meshes, textures, and etc.) as the main database to retrieve the top $k = 10$ objects that matches the asset queries. Due to noises in assets' language annotations and the extreme diversity of objects in Objaverse (e.g. many of the assets are not common household objects), object retrieved this way are potentially not suitable for the proposed task. We further use Gemini-Pro [270], a state-of-the-art vision-language

models (VLM) to verify the retrieved assets and filter out the undesired ones. (See Appendix 7.6.1 for more details for the retrieval and verification process.) In practice, we found objects retrieved this way work well for rigid object manipulation tasks. For soft-body manipulation tasks, where a more consistent and controllable target shape for the soft-body under manipulation is desired, and fine-grained details of geometry and texture are secondary, we ask GPT-4 to come up with desired target shapes, and use a text-to-image followed by image-to-mesh generation pipeline to generate the needed mesh. We use Midjourney [184] as our text-to-image generative model, and Zero-1-to-3 [163] as our image-to-mesh generative model.

Asset size Assets generated or retreived from Objaverse [50] and PartNetMobility [306] are usually not of physically plausible sizes. To account for this, we query GPT-4 to generate the sizes of the assets such that: 1) the sizes should match real-world object sizes; 2) the relative sizes between objects allow a plausible solution for solving the task, e.g., for the task of "putting a book into the drawer", the size of the drawer should be larger than the book.

Initial asset configuration For certain tasks, the articulated object should be initialized with valid states for the robot to learn the skill. For example, for the task of "close the window", the window should be initialized in an open state; similarly, for the task of "opening the door", the door should be initially closed. Again, we query GPT-4 to set the initial configurations of these articulated objects, specified in joint angles.

Scene configuration Spatial configuration specifying the location and relevant poses of each asset in the scene is crucial for both producing plausible environments and allowing valid skill learning. E.g., for the task of "retrieving a document from the safe", the document needs to be initialized `inside` the safe; for the task of "removing the knife from the chopping board", the knife needs to be initially placed `on` the chopping board. RoboGen queries GPT-4 to generate the locations for each asset as well as such special spatial relationships with the task description as the input. To avoid collision between objects, RoboGen instructs GPT-4 to place objects in a collision-free manner. (See Appendix 7.6.2 for more details.) With the generated scene components and their corresponding configurations, we populate the scene accordingly. See Figure 7.1 for a collection of example scenes and tasks generated by RoboGen. More examples of the generated scenes are available on our project

website.

7.3.3 Training Supervision Generation

To acquire the skill for solving the proposed task, supervisions for skill learning are needed. To facilitate the learning process, RoboGen first queries GPT-4 to plan and decompose the generated task into shorter-horizon sub-tasks. After the decomposition, RoboGen then queries GPT-4 to choose a proper algorithm for solving each sub-task. There are three different types of learning algorithms integrated into RoboGen: reinforcement learning [85, 238], gradient-based trajectory optimization [305, 313], and action primitive with motion planning [125]. Each of these is suited for different tasks, e.g., gradient-based trajectory optimization is more suitable for learning fine-grained manipulation tasks involving soft bodies such as shaping a dough into a target shape [156, 313]; action primitives coupled with motion planning are more reliable in solving the task such as approaching a target object via a collision-free path; reinforcement learning better suits tasks that are contact rich and involving continuous interaction with other scene components, e.g., legged locomotion, or when the required actions cannot be simply parameterized by discrete end-effector poses, e.g., turning the knob of an oven. We provide examples and let GPT-4 choose which learning algorithm to use conditioned on the generated sub-task.

We consider several action primitives including grasping, approaching and releasing a target object. Since parallel jaw gripper can be limited when grasping objects with diverse sizes, we consider a robotic manipulator equipped with a suction cup to simplify object grasping. The grasping and approaching primitives are implemented as follows: we first randomly sample a point on the target object or link, compute a gripper pose that aligns with the normal of the sampled point, and then use motion planning to find a collision-free path to reach the target gripper pose. After the pose is reached, we proceed along the normal direction until a contact is made with the target object. For the grasping and approaching primitives, RoboGen asks GPT-4 to specify the target object to grasp or approach, conditioned on the sub-task. See Appendix 7.6.3 for more implementation details about the action primitives.

For sub-tasks trained with RL, we prompt GPT-4 to write corresponding reward functions with three in-context examples. For rigid manipulation and locomotion

tasks, the reward functions are based on the low-level states which GPT-4 can query via provided simulator APIs. For soft body manipulation tasks, RoboGen uses reward functions specified as the earth-mover distance between the particles of current and target shape. We prompt GPT-4 to generate a text description of the target shape, and then use a text-to-3d model [163] to generate the mesh of the target shape using the text description, as described in Section 7.3.2.

7.3.4 Skill Learning

Once we obtained all the required information for the proposed task, including scene components and configurations, task decompositions, and training supervisions for the decomposed sub-tasks, we are able to construct the scene in simulation for the robot to learn the required skills for completing the task. For long-horizon tasks that involve multiple sub-tasks, we adopt a simple scheme of learning each sub-task sequentially: for each sub-task, we run the learning algorithm for $N = 8$ times and use the end state with the highest reward as the initial state for the next sub-task. As aforementioned, we use a combination of techniques for skill learning, including reinforcement learning, gradient-based trajectory optimization, and action primitive with motion planning, selected on the fly conditioned on the task generated. For more details, please refer to Appendix 7.6.3.

Discussion on design choices Our framework design prioritizes its foundational structure over specific backend models used in the initial implementation, and our system is agnostic to the backend LLM/VLM/generative model used, ensuring that RoboGen can be continuously improved by upgrading the backend modules with newer models once they become available. In addition, while human-designed 3D asset databases currently still present better quality, automated text-to-3D generative pipelines utilize massive 2D image resources available online and holds a better potential in further scaling up. As a result, we intentionally added support for both retrieval-based and generation-based methods for acquiring assets, and anticipate our method evolving towards a fully generative model in the future.

7.4 Experiments

RoboGen is an automated pipeline that can be queried endlessly, and generate a continuous stream of skill demonstrations for diverse tasks. Our experiments aim to answer the following questions: 1) **Task Diversity**: How diverse are the tasks proposed by RoboGen for robotic skill learning? 2) **Scene Validity**: Does RoboGen generate valid simulation environments? 3) **Training Supervision Validity**: Does RoboGen generate valid task decomposition and training supervisions for the task that will induce intended robot skills? 4) **Skill Learning**: Does integrating different learning algorithms in RoboGen improve skill learning performance? 5) **System**: Can the whole system produce diverse and meaningful robotic skill demonstrations?

7.4.1 Experimental Setup

Our proposed system is generic and agnostic to specific simulation platforms. However, since we consider a wide range of task categories ranging from rigid dynamics to soft body simulation, and also consider skill learning methods such as gradient-based trajectory optimization which necessitates a differentiable simulation platform, we used Genesis for deploying RoboGen, a simulation platform for robot learning with diverse materials and fully differentiable[1]. For skill learning, we use SAC [85] as the RL algorithm. The policy and Q networks are both Multi-layer Perceptrons (MLP) of size $[256, 256, 256]$, trained with a learning rate of $3e-4$. For each sub-task, we train with 1M environment steps. We use BIT* [71] as the motion planning algorithm, and Adam [129] for gradient-based trajectory optimization for soft body manipulation tasks. More implementation details can be found in Appendix 7.6.3.

7.4.2 Evaluation Metrics and Baselines

The following evaluation metrics and baselines are used:

Task Diversity The diversity of the generated tasks can be measured in many aspects, such as the semantic meanings of the tasks, scene configurations of the generated simulation environments, the appearances and geometries of the retrieved object

[1]Genesis is still under development and will be release publicly soon. We build our system on top of an internal version.

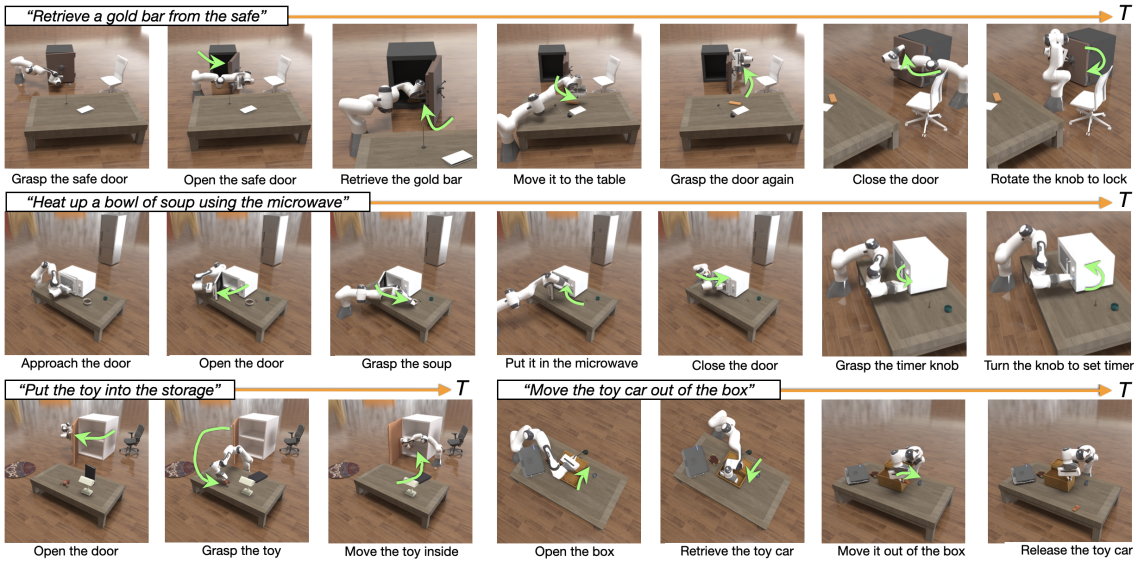


Figure 7.3: Snapshots of the learned skills on 4 example long-horizon tasks.

	RoboGen	Behavior-100	RLbench	MetaWorld	Maniskill2	GenSim
Number of Tasks	106	100	106	50	20	70
Task Description - Self-BLEU ↓	**0.284**	0.299	0.317	0.322	0.674	0.378
Task Description - Embedding Similarity (SentenceBert) ↓	**0.165**	0.210	0.200	0.263	0.194	0.288
Scene Image - Embedding Similarity (ViT) ↓	**0.193**	0.389	0.375	0.517	0.332	0.717
Scene Image - Embedding Similarity (CLIP) ↓	**0.762**	0.833	0.864	0.867	0.828	0.932

Table 7.1: **Comparison on task diversity** with representative human-designed robotics datasets Behavior-100, RLBench, MetaWorld, Maniskill2, and concurrent work GenSim [286].

assets, and the robot actions required to perform the task. For semantic meanings of the tasks, we perform quantitative evaluations by computing the Self-BLEU [204, 340] and the embedding similarity [340] on the generated task descriptions, where lower scores indicate better diversity. In addition to the semantics, we also compare the diversity of the generated tasks in the image space, measured by the embedding similarity of the rendered images of the scenes at the initial state with both ImageNet pre-trained ViT [54] and CLIP models [217]. We compare to established benchmarks, including RLBench [113], Maniskill2 [80], Meta-World [322], and Behavior-100 [258]. We also compare to concurrent work [286], which leverages LLM to write codes for generating table-top rigid object manipulation tasks. For robot actions, we evaluate RoboGen qualitatively using the generated environments and visualizations of learned robot skills.

Scene Validity To verify that the retrieved objects match the requirements of the task, we compute the BLIP-2 scores [141] between rendered images of the retrieved objects in the simulation scene, and the text descriptions of the objects. We compare with two ablations of our system. A) *w/o object verification:* We retrieve objects based on matching of language descriptions without using a VLM to verify the retrieved object. B) *w/o size verification:* We use the default size associated with the retrieved asset without quering LLM for plausible sizes. We also evaluate scene-level validity via human evaluation, examining whether the generated scenes align with the task descriptions, and if the scene configurations and retrieved objects are correct.

Training Supervision Validity We perform human verification by asking a human expert to manually inspect whether the generated decompositions and reward functions are reasonable for solving the task. We also perform qualitative evaluations by presenting videos of the learned skills using the generated decomposition and training supervisions.

Skill Learning Performance We provide quantitative analysis on the skill learning success rate. The success rate is defined as the ratio of runs that successfully learn the skill over all attempting runs for a task. In addition, we compare to an ablation where we remove the options of using motion planning-based primitive, and rely purely on reinforcement learning to learn the skills on a set of generated articulated-object manipulation tasks.

System We show qualitative evaluations of the whole system, by providing videos of over 100 learned skills on our website. Figure 7.1 includes snapshots of representative tasks. We also provide a detailed list of generated tasks along with task statistics (e.g., average number of sub-steps) and a detailed failure analysis in Appendix 7.7.1 and 7.7.3, respectively.

7.4.3 Results

Task Diversity We compare RoboGen with several established robotics benchmarks in terms of task diversity and report results in Table 7.1. Note that RoboGen can generate an endless stream of tasks when queried repeatedly, but here we evaluate a version with 106 tasks generated, comparable to prior works. RoboGen achieves the

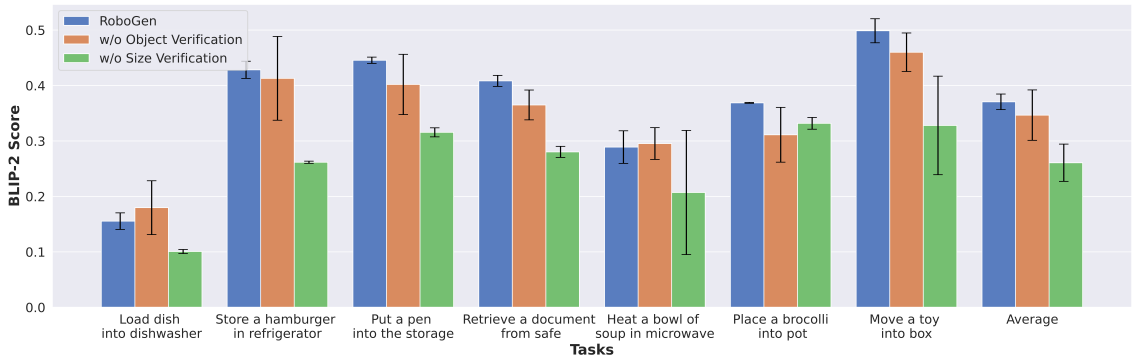


Figure 7.4: We compare the BLIP-2 score of ablations of RoboGen on 7 tasks to evaluate the importance of both object and size verification.

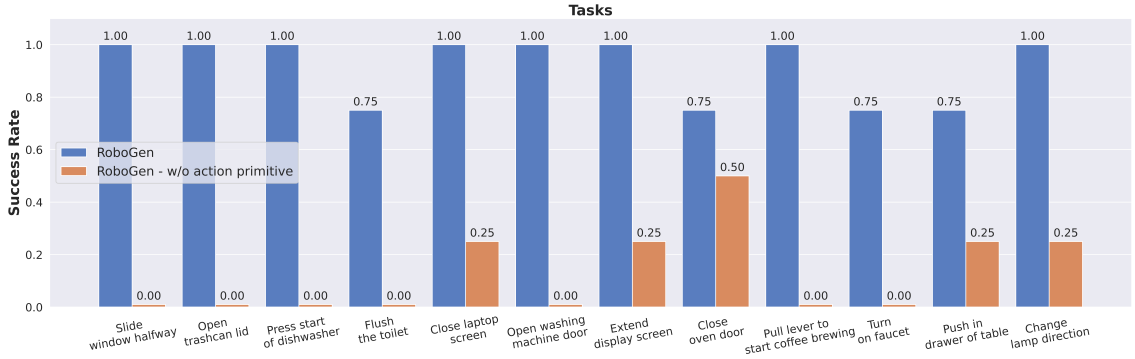


Figure 7.5: Among 12 articulated object manipulation tasks, the success rate decreases drastically if only RL is used for skill learning.

lowest Self-BLEU, as well as the lowest similarity score in both language and image space, demonstrating that our pipeline can generate tasks whose semantic and visual diversity matches or surpasses prior manually crafted skill learning benchmarks and datasets. We also note the diversity of scenes and tasks generated by RoboGen is noticeably higher than GenSim [286]. We believe part of the reason is that GenSim only generates table-top pick-and-place manipulation tasks with a small number of assets from the Ravens benchmarking dataset [248]. In contrast, RoboGen can generate a broader range of tasks such as articulated object manipulation tasks that reason about their affordances and functionalities, legged locomotion, and soft body manipulation tasks, meanwhile leverage more diverse assets retrieved from open-world databases such as Objaverse, resulting in much higher diversity in both task semantics

and scene images. We also provide the full list of generated tasks, including the task name and task descriptions in Appendix 7.7.1, and refer readers to our project website for visualizations of the generated tasks.

Scene Validity Figure 7.4 shows the BLIP-2 score of all compared methods on an example set of 7 generated tasks. As shown, removing the size verification leads to drastic decrease in BLIP-2 score. This is expected as the default asset sizes can be drastically different from plausible real-world sizes. The ablation "w/o object verification" also has a lower BLIP-2 score and a larger variances, indicating our verification step improves validity of the constructed scene. The results demonstrate the importance of using both object and size verification in RoboGen. In addition, we conducted manual evaluations of the generated tasks for scene-level validity. Out of 155 generated tasks (full list in Appendix 7.7.1), we found 13 failures due to incorrect scene generation. The failures can be categorized into 1) required functionality not supported by the assets, e.g., loading paper into a printer asset which do not have a movable tray. 2) incorrect semantic understanding of articulated object's joint state, i.e., failure to correctly map the joint angle value of an articulated object to its semantic state, e.g., an LLM cannot judge whether the joint angle value 0 corresponds to the door being opened or closed. 3) failure to find matched assets for tasks that require extremely precise spatial relationships, e.g., it is hard to retrieve or generate stapler and staples whose size and geometry exactly match each other for the task of loading the staples into the stapler. We provide a detailed analysis in Appendix 7.7.3 on the failure cases and potential solutions to address them in future work.

Training Supervision Validity Figure 7.3 demonstrates the skills learned with the generated training supervisions from RoboGen, i.e., the task decompositions and reward functions, on 4 example *long-horizon* tasks. As shown, the robot successfully learns skills to complete the corresponding tasks, suggesting that the automatically generated training supervisions are effective in deriving meaningful and useful skills. We also manually inspected the generated decompositions and reward functions, and found 6 failure cases in the 155 generated object manipulation tasks. The errors can be categorized into 1) referring to undefined variables; 2) reward does not encode the intended behavior. Examples include incorrect semantic understanding of articulated object state, e.g., the task is to fold the chair, yet the generated reward actually encourages unfolding the chair due to misunderstanding of the mapping between

joint angle values and object state. We also find it hard to generate correct rewards for continuous motions such as "moving robotic hand back-and-forth", or "knock the door". Again, see Appendix 7.7.3 for detailed failure analysis and discussion on potential solutions.

Skill Learning We first evaluate the success rate of our skill learning pipeline on a subset of 50 generated object manipulation tasks, 7 soft-body manipulation tasks, and 12 locomotion tasks. Over all 69 benchmarked tasks, RoboGen achieves an average success rate of 0.774, indicating 3 out of 4 runs could lead to successful skill learning. Detailed statistics of the tasks are available in Appendix 7.7.2.

Further, we compare to an ablated version of RoboGen where only RL is used for skill learning. We randomly select 12 tasks that involve interactions with articulated objects for this comparison. The results are shown in Figure 7.5. As shown, allowing RoboGen to select the optimal learning algorithms beneficial for achieving higher performance for completing the tasks. When only RL is used, the skill learning completely fails for most tasks.

System Figure 7.1 and 7.3 show some representative tasks and learned skills generated by RoboGen. As shown in Figure 7.1, RoboGen can generate diverse tasks for skill learning spanning rigid/articulated object manipulation, legged locomotion and soft body manipulation. Figure 7.3 further shows that RoboGen is able to deliver long-horizon manipulation skills with reasonable decompositions. For extensive qualitative results of proposed tasks and learned skills, please refer to our project site. Again, please refer to Appendix 7.7 for a list of generated tasks, their statistics, and a detailed failure analysis.

7.5 Conclusion & Limitations

We introduced *RoboGen*, a generative agent that automatically proposes and learns diverse robotic skills at scale via generative simulation. RoboGen utilizes the latest advancements in foundation models to automatically generate diverse tasks, scenes, and training supervisions in simulation, making a foundational step towards scalable robotic skill learning in simulation, while requiring minimal human supervision once deployed. Our system is a fully generative pipeline that can be queried endlessly, producing a large number of skill demonstrations associated with diverse tasks

and environments. Our current system still has several limitations: 1) Large-scale verification of learned skills is still a challenge in the current pipeline, which could potentially be addressed by incorporating feedback from multi-modal foundation models in the future. 2) Our paradigm is intrinsically constrained by sim-to-real gaps for real-world deployment, which is a stand-alone research field. However, given the recent rapid advancements in physically accurate simulation [142] and techniques like domain randomization [271, 313] and realistic sensory signal rendering [334], we anticipate a continual narrowing of this gap in the near future.

7.6 Additional Implementation Details

7.6.1 Asset Retrieval and Verification

For each object in Objaverse, we obtain a list of language descriptions of it by combining the default annotations and a more cleaned version of annotations from [170]. Given the language description of the asset we want to retrieve, we use Sentence-Bert [223] to get the embedding of the description, and retrieve k objects from Objaverse whose language embeddings are the most similar to the language embedding of the target asset. Due to noises in the object annotations, there can be significant discrepancies between the actual asset and the intended target, even when the similarity score in the language embedding space is high. To resolve this, we further use Gemini-Pro [270] a state-of-the-art vision-language model (VLM) to verify the retrieved assets and filter out the undesired ones. Specifically, we input an image of the retrieved object to the VLM mode to generate a caption of the object. The caption, together with the description of the desired asset and the description of the task, are fed back into GPT-4 to verify if the retrieved asset is appropriate to be used in the proposed task.

7.6.2 Collision Resolving in Scene Generation

When the LLM generate the initial pose of the objects, we prompt it to leverage its basic spatial understanding and tries to place the objects in different locations. We use this as the initialization, and check potential collisions in the initial scene

configuration. For any detected collision between two objects, we identify the collision node of the objects in contact, and push their center of mass away along the opposite directions of the collision normals to resolve collision.

7.6.3 Skill Learning

For reinforcement learning, we use SAC [85] as the RL algorithm. For object manipulation tasks, the observation space is the low-level state of the objects and robot in the task. The policy and Q networks used in SAC are both Multi-layer Perceptrons (MLP) of size $[256, 256, 256]$. We use a learning rate of $3e-4$ for the actor, the critic, and the entropy regularizer. The horizon of all manipulation tasks are 100, with a frameskip of 2. The action of the RL policy is 6d: where the first 3 elements determines the translation, either as delta translation or target location (suggested by GPT-4), and the second 3 elements determines the delta rotation, expressed as delta-axis angle in the gripper's local frame. For each sub-task, we train with 1M environment steps. For locomotion tasks, the cross entropy method (CEM (De Boer et al., 2005)) is used for skill learning, which we find to be more stable and efficient than RL. The ground-truth simulator is used as the dynamcis model in CEM, and the actions to be optimized are the joint angle values of the robot. The horizon for all locomotion tasks are 150, with a frameskip of 4

For action primitives, we use BIT* [71] implemented in the Open Motion Planning Library (OMPL) [265] as the motion planning algorithm. For the grasping and the approaching primitive, we first sample a surface point on the target object or link, then compute a gripper pose that aligns the gripper y axis with the normal of the sampled point. The pre-contact gripper pose is set to be 0.03m above the surface point along the normal direction. Motion planning is then used to find a collision-free path to reach the target gripper pose. After the target gripper pose is reached, we keep moving the gripper along the normal until contact is made.

For soft body manipulation tasks, we use Adam [129] for gradient-based trajectory optimization. We run trajectory optimization for 300 gradient steps. We use a learning rate of 0.05 for the optimizer. The horizons of all manipulation tasks are either 150 or 200. We use Earth Mover's distance between object's current and target shape as the cost function for trajectory optimization.

For querying GPT-4, we used a temperature between $0.8 - 1.0$ for task proposal to ensure diversity in the generated tasks. For all other stages of RoboGen, we use temperature values between $0 - 0.3$ to ensure more robust responses from GPT-4.

7.7 Additional Details on Generated tasks, Statistics, and Analysis

7.7.1 List of Tasks and Statistics

Note that RoboGen can be used to generate different type of tasks including rigid and articulated object manipulation, soft body object manipulation, and legged locomotion, but the major diversity of the tasks lies in manipulating articulated and rigid objects in the current framework, due to the varied nature of these objects in everyday life.

Table 7.2 provides the list of rigid and articulated object manipulation tasks that are generated using RoboGen at the time of submission. We note that RoboGen can be constantly queried to generate more tasks. Figure 7.6 shows the distribution of number of substeps for these generated tasks. As shown, most tasks are short-horizon and can be solved within 4 substeps. Longer-horizon tasks require 8 and up to 10 substeps to solve. The average number of substeps for all tasks is 3.13. Figure 7.6 also presents the distribution of substeps to be solved using RL or motion planning based primitives. Please refer to the caption of the figure for more details.

Table 7.3 shows a list of representative soft body manipulation tasks that RoboGen generates, and Table 7.4 shows the a list of example generated locomotion tasks.

Table 7.2: List of generated tasks.

Task name	Task description	# of sub-steps	# of RL substeps	# of primitive substeps
Rotate Laptop Screen	The robot arm rotates the laptop screen to a certain angle for better view	2	1	1
Move Laptop	The robot arm lifts and moves the laptop to a new location	3	2	1

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of substeps	# of RL substeps	# of primitive substeps
Close Laptop Lid	The robotic arm will close the laptop lid	2	1	1
Open Laptop Lid	The robotic arm will open the laptop lid	2	1	1
Pack Item In Suitcase	The robot arm places an item .for example, a folded shirt. inside the suitcase	4	2	2
Extend Suitcase Handle	The robotic arm will extend the suitcases handle in order to pull or push the suitcase	2	1	1
Pull Suitcase on Wheels	The robot arm extends the suitcase handle, grips it in a way to let the suitcase stand on its wheels and pulls it	3	2	1
Lift Suitcase	The robotic arm will lift the suitcase by its handle	2	1	1
Partially Close Window	The robotic arm partially closes one of the slider translation windows	2	1	1
Open Window Halfway	The robotic arm will open one of the slider translation windows halfway to let fresh air in	2	1	1
Fully Open Window	The robotic arm will open both of the slider translation windows to their full extent for maximum ventilation	4	2	2
Close Window	The robotic arm closes both slider translation windows	4	2	2
Open and Close Toilet Lid	The robot arm will interact with the hinge lid of the toilet to first open it and then close it	4	2	2
Open and Close Toilet Pump Lid	The robot arm will interact with the slider pump lid to first open it and then close it	3	2	1
Flush the Toilet	The robotic arm will interact with the hinge lever of the toilet to flush it	3	1	2
Set Clock Time	The robotic arm adjusts the hinge hands of the clock to set the desired time	6	2	4
Move Clock Ahead for Daylight Saving	The robotic arm moves the clock hands ahead by 1 hour to adjust for daylight saving	2	1	1

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of sub-steps	# of RL substeps	# of primitive substeps
Move Clock Back at End of Daylight Saving	The robot arm moves the clock hands back by 1 hour to adjust to the end of daylight saving	2	1	1
close the oven door	The robot arm needs to close the oven door after use This task involves moving towards the oven door and applying force to close it	2	1	1
Extend Display Screen	The robotic arm will extend the slider translation screen to enlarge the display	2	1	1
Retract Display Screen	The robotic arm will retract the slider translation screen to make the display smaller	2	1	1
Adjust Display Angle	The robotic arm adjusts the display base link to change the viewing angle	2	1	1
Rotate Display Base	The robotic arm will rotate the display base to point the display to a different direction	2	1	1
Rinse a Plate	The robot arm holds a plate under the spout, turns on the faucet to rinse the plate, then turns off the faucet	8	3	5
Turn On Faucet	The robotic arm operates the hinge switch of the faucet in order for water to flow from the spout	2	1	1
Wash Hands	The robot arm acts as if its washing hands to demonstrate good hygiene	8	4	4
Fill a Glass of Water	The robot arm first turns on the faucet, waits for a glass to fill, then turns off the faucet	5	3	2
Fold Chair	The robotic arm will fold the chair to save room or for easy carrying	3	1	2
Position Chair for Seating	The robotic arm positions the unfolded chair in a desired location for a person to sit	3	1	2
Unfold Chair	The robotic arm will unfold the folding chair to make it suitable for sitting	3	1	2
Lift Chair	The robotic arm lifts the chair from the ground to place it into another location	4	2	2

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of sub-steps	# of RL substeps	# of primitive substeps
Staple Papers	The robot arm gathers a few loose sheets of paper and uses the stapler to staple them together	6	2	4
Close Stapler Lid	The robot arm closes the lid of the stapler after it has been opened	2	1	1
Open Stapler Lid	The robotic arm will open the lid of the stapler	2	1	1
Load Staples into Stapler	The robot arm inserts new staples into the stapler	6	3	3
Turn On the Printer	The robot arm pushes the slider button to turn on the printer	2	1	1
Load Paper into Printer	The robot arm loads paper into the printer via the input tray, typically located on the printer body	2	1	1
Print a Document	The robot interacts with the printer to print a document The robot arm first places a document on the printer, then moves the button to initiate the print	4	2	2
Stop a Printer	The robot arm stops a printer by moving the slider button to the stop position	2	1	1
Fill Kettle with Water	The robot arm opens the kettle lid, holds a water jug to fill the kettle with water, and then closes the lid	6	3	3
Pour Water from Kettle	The robot arm holds the kettle handle, tilts the kettle to pour water into a cup	4	2	2
Open Kettle Lid	The robotic arm will open the kettle lid	2	1	1
Lift Kettle by Handle	The robotic arm will lift the kettle by its handle	2	1	1
close the drawer of the table	The robot arm will close the drawer of the table	2	1	1
Close Door	The robotic arm will close the door	2	1	1
Knock On Door	The robotic arm will knock on the door in a typical way a human would	3	3	0
Partially Open Door	Open the door partially for ventilation or for casual conversation without fully opening it	2	1	1
Open Door	The robotic arm will open the door	2	1	1

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of sub-steps	# of RL substeps	# of primitive substeps
Open Partial Box Lid	The robotic arm will partially open the box lid based on certain degree, to demonstrate kinematic control	2	1	1
Store an Object Inside Box	The robot arm places a small object inside the box and closes the lid	6	3	3
Open Box Lid	The robotic arm will open the box lid	2	1	1
Retrieve an Object From Box	The robot arm opens the box lid, takes a small object from the box, and then closes the lid	6	3	3
Push Drawer In	After retrieving an item from the drawer, the robot arm slides the drawer back into the box	2	1	1
Close Box Lid	The robotic arm closes the lid of the box	2	1	1
Pull Drawer Out	The robotic arm uses the prismatic joint to slide the drawer out from the box	2	1	1
Making Coffee	The robot arm opens the lid of the container, places coffee grounds inside, then closes the lid and starts the brewing process by adjusting the knob	8	4	4
Turning On Coffee Machine	The robotic arm will adjust the hinge knob on the coffee machine to the on setting	2	1	1
Change Cleaning Cycle	Robot changes the cleaning cycle of the dishwasher by interacting with one of the slider buttons	2	1	1
Open Dishwasher Door	The robotic arm will open the dishwasher door	2	1	1
Load Dishwasher	Robot arm places a plate inside the dishwasher	6	3	3
Press Start Button	The robot will press the start button on the dishwasher to begin the washing cycle	3	1	2
Close Dispenser Lid	After filling or extracting contents, the robotic arm will close the lid of the dispenser	2	1	1
Extract Contents	The robot arm will open the dispenser lid and proceed to extract the contents inside the dispenser	6	3	3
Open Dispenser Lid	The robotic arm will open the lid of the dispenser	2	1	1

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of substeps	# of RL substeps	# of primitive substeps
Fill Dispenser	The robotic arm opens the dispenser lid and then pours the desired content into the dispenser	5	3	2
Rotate Fan Rotor	The robotic arm will apply a force to the rotor of the fan, causing it to rotate	3	1	2
Change Fan Direction	The robotic arm will change the direction of the fan by physically moving the entire fan	2	1	1
Position Fan To Cool Off a Room	The robot arm moves the fan to a location in order to cool off a specific area in a room	2	1	1
Turn Off Water Faucet	The robotic arm will rotate the switch of the faucet to cut off the water supply	2	1	1
Angle Laptop Screen	The robot positions the laptop screen to a desired angle for better visibility	2	1	1
Opening Refrigerator Door	The robotic arm will open one of the refrigerator doors	2	1	1
Opening Both Refrigerator Doors	The robotic arm opens both the refrigerator doors one after the other	4	2	2
Load item into the refrigerator	The robotic arm will open one of the refrigerator doors, place an item inside, and close the door	6	3	3
Retrieving an item from the refrigerator	The robotic arm will open one of the refrigerator doors, retrieve an item, and then close the door	6	3	3
Dispose Toilet Paper into Toilet	A robotic arm picks up a piece of toilet paper and disposes of it in the toilet by dropping it in and then closing the lid	10	3	7
Close Trashcan Lid	The robotic arm will close the trashcans lid	2	1	1
Open Trashcan Lid	The robotic arm will open the trashcans lid	2	1	1
Move the Trashcan	The robot arm pushes the trashcan from one place to another	2	1	1
Change Lamp Direction	The robotic arm will alter the lamp's light direction by manipulating the lamps head	2	1	1
Rotate Lamp Base	The robot arm will rotate the lamp base to adjust the lamps general orientation	2	1	1

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of sub-steps	# of RL substeps	# of primitive substeps
Adjust Lamp Position	The robotic arm will adjust the position of the lamp using its hinge rotation bars, enabling the robot to direct the lamps light to a specific area	6	3	3
Change Lamp Direction	The robotic arm will alter the lamp's light direction by manipulating the lamps head	2	1	1
Close Drawer	The robotic arm will push the drawer closed	2	1	1
Retrieve Object from Drawer	The robot arm opens the drawer, retrieves an object from inside, and then closes the drawer	6	3	3
Open Drawer	The robotic arm will pull the drawer open	2	1	1
Store Object in Table Drawer	The robot arm puts an item, like a book, into a drawer in the table	6	3	3
Throw Trash Away	The robotic arm places an item of trash inside the trash can	7	3	4
Insert New Trash Bag	The robotic arm inserts a new trash bag into the trash can	5	3	2
Check Contents of the Pot	The robot arm slides the lid of the pot to check the contents inside the pot	3	1	2
Stir Contents in Pot	The robot arm removes the lid of the pot and stirs the pots contents with a stirring spoon	4	2	2
Remove Pot Lid	The robotic arm will slide the lid of the pot aside	3	1	2
Select Washing Cycle	The robotic arm will push one of the washing machines slider buttons to select a washing cycle	2	1	1
Load Clothes Into Washing Machine	The robot arm opens the washing machine door and places clothes inside	4	2	2
Adjust Washing Settings	The robot arm rotates a knob to adjust washing settings such as temperature or spin speed	2	1	1
Open Washing Machine Door	The robotic arm will open the washing machine door	2	1	1
Move Door Slightly Open	The robotic arm opens the door slightly to allow for some air circulation without fully opening it	3	1	2

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of substeps	# of RL substeps	# of primitive substeps
Deliver an Object	The robot arm holds an object, opens the door, passes through, then closes the door behind it This represents the robot arm delivering an object from one room to another	8	3	5
Find Door Position	The robot arm would touch different parts of the door to find its initial position It is useful to know the initial position for actions like opening or closing	4	2	2
Regulate Coffee Strength	The robot arm rotates a knob to adjust the strength of the coffee	2	1	1
Insert Portafilter	The robot arm inserts the portafilter into the coffee machine	3	2	1
Adjust Machine Settings	The robot arm adjusts a knob to alter machine settings	2	1	1
Pull Lever to Start Coffee Brewing	The robot arm pulls a lever to start the brewing process of the coffee machine	2	1	1
Steam Milk	The robot operates a lever to steam milk for the coffee	3	2	1
Unload Dishes from Dishwasher	The robot arm retrieves clean dishes from the dishwasher	6	3	3
Start Dishwasher Cycle	The robot arm turns the dishwasher knob to start the washing cycle	2	1	1
Open Dishwasher Door	The robotic arm will open the dishwasher door for placing or removing dishes	2	1	1
Straighten Display Screen	The robotic arm will straighten the display screen if it has been tilted or rotated	3	1	2
Tilt Display Screen	The robotic arm will tilt the display screen to adjust viewing angle	3	1	2
Position Display Screen	The robotic arm will move the display screen to a desired location	2	1	1
Orient Globe Towards Specific Country	The robot arm rotates the globe such that a specific country on the globes surface faces the viewer	2	1	1
Rotate Globe Horizontally	The robotic arm will rotate the globe horizontally to display various continents and countries on its surface	2	1	1

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of substeps	# of RL substeps	# of primitive substeps
Spin Globe Gently for Leisure	The robot arm spins the globe gently, as a relaxing activity or a playful interaction	2	1	1
Adjust Lamp Height	The robot arm will adjust the height of the lamp by manipulating the rotation bars	6	3	3
Turn On Lamp	The robotic arm turns on the lamp by pressing the toggle button	3	1	2
Set Soup Bowl in Microwave	The robot arm will set a bowl of soup on the microwaves rotation tray and set the timer	7	3	4
Rotate Power Knob	The robotic arm rotates the power know to set the heating power level	2	1	1
Press Microwave Button	The robot arm slides the microwave button	3	1	2
Set Timer	The robotic arm rotates the timer knob to set the duration for heating	2	1	1
Open Microwave Door	The robotic arm will open the microwave door	2	1	1
Open Oven Door	The robot arm is programmed to open the door of the oven	2	1	1
Adjust Oven Timer	The robot arm is to manipulate one of the ovens hinge knobs to set an appropriate timer	2	1	1
Set Oven Temperature	The robot arm is to adjust another knob to set the appropriate temperature for cooking	2	1	1
Set Oven Function	The robot arm needs to adjust another knob to set the desired oven function – for example, circulating air, grilling or bottom heat	2	1	1
Open Fridges Freezer Door	The robot arm opens the freezer compartment door of the refrigerator	2	1	1
Move Cart Forward	The robotic arm will push the cart forward	2	1	1
Turn Cart	The robotic arm will turn the cart to change its direction	2	1	1
Load Object onto Cart	The robot arm places an object onto the cart	3	1	2
Unload Object from Cart	The robot arm takes an object off from the cart	3	1	2

Continued on next page

Table 7.2 continued from previous page

Task name	Task description	# of substeps	# of RL substeps	# of primitive substeps
Adjust Chair Height	The robotic arm will adjust the height of the chair by interacting with the knob	2	1	1
Move Chair	The robot arm will move the chair using the wheels	2	1	1
Rotate Chair	The robot arm rotates the chair to a desired direction	2	1	1
Tilt Chair Seat	The robot arm tilts the chair seat to a desired angle	2	1	1
Open Eyeglasses	The robotic arm will unfold the legs of the eyeglasses	4	2	2
Place Eyeglasses on Table	The robot arm picks up the eyeglasses and places them on a table	3	1	2
Store an Item in Safe	The robot arm opens the safe, places an item inside, and then closes and locks the safe	8	4	4
Turn Safe Knob	The robotic arm will turn one of the safes knobs to unlock it	2	1	1
Retrieve an Item from Safe	The robot arm unlocks the safe, opens the door, retrieves an item from inside, and then closes and locks the safe	8	4	4
Open Safe Door	The robotic arm will open the safe door	2	1	1
Open Trashcan Lid	The robotic arm will open the lid of the trashcan	2	1	1
Open Dispenser Lid	The robotic arm will open the lid of the dispenser	2	1	1
Turn On Water Faucet	The robotic arm will rotate the switch of the faucet to turn on the water	2	1	1
Open Laptop	The robotic arm opens the unfolded state of the laptops screen	2	1	1
Open Toilet Lid	The robotic arm will carefully open the lid of the toilet	2	1	1
Close Dispenser Lid	The robotic arm will close the dispenser lid after use	2	1	1
Close Table Drawer	The robotic arm will close the open drawer on the table	2	1	1
Open Trash Can	The robotic arm will open the trash can lid	2	1	1
Close Toilet Lid	The robotic arm will put down the lid of the toilet	3	1	2

Continued on next page

Task Name	Task Description
Bend the noodle into a U shape	The robot needs to to bend an initial straight noodle into the shape of letter "U"
Flatten the rice dough	The robot uses a square dough flattener to flatten a rice dough
Cut dough in half	The robot uses a knife to cut a dough in half
Shape dough	The robot uses two square dough flatteners to shape the dough into a baguette
Lift up dumping	The robot uses two square dough flatteners to lift up a dumpling
Roll out dough	The robot uses a rolling pin to flatten a dough
Put filling onto wrapper	The robot needs to grasp a dumpling filling and put it on top of the dumpling wrapper

Table 7.3: List of soft body manipulation tasks RoboGen generated.

Task Name	Task Description
Jump forward	The legged robot needs to do a jump forward
Spin counter-clockwise	The legged robot needs to spin itself counter-clockwise around the vertical axis
Run forward	The legged robot needs to do fun forward at a high speed
Spin left without using right hind leg	The legged robot needs to spin itself to the left while not letting the right hind leg touch the ground
Jump higher than 5 meters	The legged robot needs to jump and reach a height higher than 5 meters
Flip forward	The legged robot makes a flip forward
Climb up stairs	The legged robot climbs up a staircase in the environment
Kick the soccer ball to the left	The legged robot needs to kick the soccer ball and make it move to the left
Walk backwards	The legged robot needs to move backwards
Push Ball	The legged robot needs to push the ball forward
Turn Right	The legged robot needs to turn itself to face right
Crawl forward	The legged robot needs to move forward while keeping the body in a low position

Table 7.4: List of locomotion tasks RoboGen generated.

Table 7.2 continued from previous page

Task name	Task description	# of sub-steps	# of RL substeps	# of primitive substeps
Open Door	The robotic arm will open the door by rotating the hinge	2	1	1
Turn Off Faucet	The robotic arm turns off the faucet by rotating one of the hinge switches	2	1	1
Close Window	The robotic arm will close the window to preserve indoor temperature	2	1	1
Open Box	The robot arm opens the box by manipulating the hinge lid	2	1	1
Rotate Clock Hands	Rotate the minute and hour hands of the clock with the robotic arm, simulating the passing of time	4	2	2
Unfold the Chair	The robotic arm will unfold the chair to prepare it for use	4	2	2
Open Kettle Lid	The robot arm lifts the kettle lid	2	1	1

7.7.2 Skill Learning Success Rate

Due to the randomness in the skill learning process (sampling is used in the motion planning-based action primitive, and RL inherently has randomness during exploration and training), we also provide quantitative analysis on the skill learning success rate,

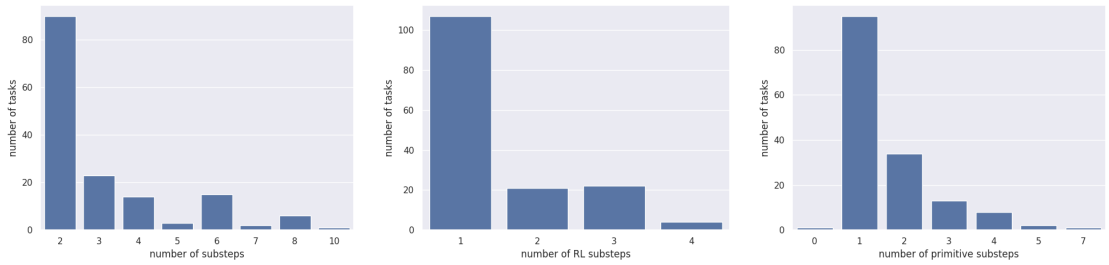


Figure 7.6: Left: The distribution of number of substeps for the generated rigid and articulated object manipulation tasks in Table 7.2. The average number of substeps is 3.13. Middle: The distribution of number of substeps that need to be solved using RL for the generated tasks. The average number of RL substeps is 1.5. Right: The distribution of number of substeps that need to be solved using motion planning based primitives for the generated tasks. The average number of such kind of substeps is 1.63. Regarding duration for solving the task: if the task's subgoals can all be solved via planning, typically each task can be solved within 10 minutes. If certain subgoals require RL to solve, it usually takes around 2-3 hours for each RL-necessary step, and the total duration thus depends on both the number and nature of the subtasks. Taking these into account, a task typically takes 4-5 hours on average. This is done using 8 threads of a CPU running at 2.5Ghz, meaning that each single node in a cluster with a 32-core (64 threads) CPU could run 8 jobs in parallel at the same time.

i.e., given a generated task with correct training supervisions, if we run the skill learning pipeline for multiple times, how many of the runs would succeed in learning the skill. The success in learning a skill is determined by a human evaluator watching the video of the learned policy.

We present the detailed skill learning success rate of 50 articulated object manipulation tasks in Table 7.5. The average skill learning success rate among these tasks is 0.745. We also benchmark the success rate for the soft-body manipulation and locomotion tasks, shown in Table 7.6 and Table 7.7.

Table 7.5: Skill learning success rate on 50 articulated object manipulation tasks.

Task name	Task description	Skill Learning Success Rate
Rotate Laptop Screen	The robot arm rotates the laptop screen to a certain angle for better view	1.0
Extend Suitcase Handle	The robotic arm will extend the suitcases handle in order to pull or push the suitcase	1.0
Open Window Halfway	The robotic arm will open one of the slider translation windows halfway to let fresh air in	1.0
Flush the Toilet	The robotic arm will interact with the hinge lever of the toilet to flush it	0.67
Move Clock Ahead for Daylight Saving	The robotic arm moves the clock hands ahead by 1 hour to adjust for daylight saving	0.38
close the oven door	The robot arm needs to close the oven door after use This task involves moving towards the oven door and applying force to close it	0.83
Open Trashcan Lid	The robotic arm will open the lid of the trashcan	1.0
Extend Display Screen	The robotic arm will extend the slider translation screen to enlarge the display	1.0
Turn On Faucet	The robotic arm operates the hinge switch of the faucet in order for water to flow from the spout	0.83
Unfold Chair	The robotic arm will unfold the folding chair to make it suitable for sitting	0.5

Continued on next page

Table 7.5 continued from previous page

Task name	Task description	Skill Learning Success Rate
Open Stapler Lid	The robotic arm will open the lid of the stapler	0.5
Turn On the Printer	The robot arm pushes the slider button to turn on the printer	1.0
Lift Kettle by Handle	The robotic arm will lift the kettle by its handle	0.83
close the drawer of the table	The robot arm will close the drawer of the table	0.75
Close Door	The robotic arm will close the door	0.5
Open Partial Box Lid	The robotic arm will partially open the box lid based on certain degree, to demonstrate kinematic control	0.83
Pull Drawer Out	The robotic arm uses the prismatic joint to slide the drawer out from the box	0.67
Turning On Coffee Machine	The robotic arm will adjust the hinge knob on the coffee machine to the on setting	0.5
Press Start Button	The robot will press the start button on the dishwasher to begin the washing cycle	1.0
Close Dispenser Lid	After filling or extracting contents, the robotic arm will close the lid of the dispenser	0.25
Open Dispenser Lid	The robotic arm will open the lid of the dispenser	0.0
Rotate Fan Rotor	The robotic arm will apply a force to the rotor of the fan, causing it to rotate	1.0
Turn On Water Faucet	The robotic arm will rotate the switch of the faucet to turn on the water	1.0
Open Laptop	The robotic arm opens the unfolded state of the laptops screen	0.8
Opening Both Refrigerator Doors	The robotic arm opens both the refrigerator doors one after the other	0.8
Open Toilet Lid	The robotic arm will carefully open the lid of the toilet	1.0
Close Trashcan Lid	The robotic arm will close the trashcans lid	0.33
Change Lamp Direction	The robotic arm will alter the lamp's light direction by manipulating the lamps head	1.0
Partially Close Window	The robotic arm partially closes one of the slider translation windows	0.5

Continued on next page

Table 7.5 continued from previous page

Task name	Task description	Skill Learning Success Rate
Close Dispenser Lid	The robotic arm will close the dispenser lid after use	1.0
Open Drawer	The robotic arm will pull the drawer open	0.75
Close Table Drawer	The robotic arm will close the open drawer on the table	0.75
Open Trash Can	The robotic arm will open the trash can lid	1.0
Remove Pot Lid	The robotic arm will slide the lid of the pot aside	1.0
Close Toilet Lid	The robotic arm will put down the lid of the toilet	1.0
Open Washing Machine Door	The robotic arm will open the washing machine door	1.0
Move Door Slightly Open	The robotic arm opens the door slightly to allow for some air circulation without fully opening it	0.67
Open Door	The robotic arm will open the door by rotating the hinge	0.5
Turn Off Faucet	The robotic arm turns off the faucet by rotating one of the hinge switches	0.67
Close Window	The robotic arm will close the window to preserve indoor temperature	0.8
Open Box	The robot arm opens the box by manipulating the hinge lid	0.8
Rotate Clock Hands	Rotate the minute and hour hands of the clock with the robotic arm, simulating the passing of time	0.4
Pull Lever to Start Coffee Brewing	The robot arm pulls a lever to start the brewing process of the coffee machine	1.0
Open Dishwasher Door	The robotic arm will open the dishwasher door for placing or removing dishes	0.6
Tilt Display Screen	The robotic arm will tilt the display screen to adjust viewing angle	0.6
Unfold the Chair	The robotic arm will unfold the chair to prepare it for use	1.0
Rotate Globe Horizontally	The robotic arm will rotate the globe horizontally to display various continents and countries on its surface	1.0
Turn On Lamp	The robotic arm turns on the lamp by pressing the toggle button	0.25

Continued on next page

Table 7.5 continued from previous page

Task name	Task description	Skill Learning Success Rate
Open Kettle Lid	The robot arm lifts the kettle lid	0.75
Open Microwave Door	The robotic arm will open the microwave door	0.25

Table 7.6: Skill learning success rate on 7 soft-body manipulation tasks.

Task name	Task description	Skill Learning Success Rate
Bend the noodle into a U shape	The robot needs to to bend an initial straight noodle into the shape of letter "U"	0.8
Flatten the rice dough	The robot uses a square dough flattener to flatten a rice dough	1.0
Cut dough in half	The robot uses a knife to cut a dough in half	1.0
Shape dough	The robot uses two square dough flatteners to shape the dough into a baguette	0.6
Lift up dumping	The robot uses two square dough flatteners to lift up a dumpling	1.0
Roll out dough	The robot uses a rolling pin to flatten a dough	1.0
Put filling onto wrapper	The robot needs to grasp a dumpling filling and put it on top of the dumpling wrapper	0.8

Table 7.7: Skill learning success rate on 12 locomotion tasks.

Task name	Task description	Skill Learning Success Rate
Jump forward	The legged robot needs to do a jump forward	1.0
Spin counter-clockwise	The legged robot needs to spin itself counter-clockwise around the vertical axis	1.0
Run forward	The legged robot needs to do fun forward at a high speed	0.6
Spin left without using right hind leg	The legged robot needs to spin itself to the left while not letting the right hind leg touch the ground	0.8

Continued on next page

Table 7.7 continued from previous page

Task name	Task description	Skill Learning Success Rate
Jump higher than 5 meters	The legged robot needs to jump and reach a height higher than 5 meters	1.0
Flip forward	The legged robot makes a flip forward	0.6
Climb up stairs	The legged robot climbs up a staircase in the environment	0.4
Kick the soccer ball to the left	The legged robot needs to kick the soccer ball and make it move to the left	0.8
Walk backwards	The legged robot needs to move backwards	1.0
Push Ball	The legged robot needs to push the ball forward	1.0
Turn Right	The legged robot needs to turn itself to face right	1.0
Crawl forward	The legged robot needs to move forward while keeping the body in a low position	0.8

7.7.3 Failure Analysis

Through manual inspection on the 155 generated tasks in Table 7.2, we found 19 failure cases in total, due to either error in the generated scene or the generated training supervisions. Table 7.8 provides a detailed analysis on the failure cases.

Among the 19 failure cases, 13 failures can be attributed to incorrect scene generation. The failures can be categorized into 1) required functionality not supported by the assets, e.g., loading paper into a printer asset which do not have a movable tray. 2) incorrect semantic understanding of articulated object's joint state, i.e., failure to correctly map the joint angle value of an articulated object to its semantic state, e.g., an LLM cannot judge whether the joint angle value 0 corresponds to the door being opened or closed. 3) failure to find matched assets for tasks that require extremely precise spatial relationships, e.g., it's hard to retrieve or generate stapler and staples whose size and geometry exactly match each other for the task of loading the staples into the stapler. Some of these failures can be addressed with additional checks, e.g., using a vision language model to verify the mapping between the joint angle values and the semantic state of the asset, while others (generate assets with required functionalities, or pair of matched assets) might require more fundamental

research to address.

6 failures are caused by incorrect reward generation. The errors can be categorized into 1) referring to undefined variables; 2) reward does not encode the intended behavior. Examples include incorrect semantic understanding of articulated object state, e.g., the task is to fold the chair, yet the generated reward actually encourages unfolding the chair due to misunderstanding of the mapping between joint angle values and object state. We also find it hard to generate correct rewards for continuous motions such as "moving robotic hand back-and-forth", or "knock the door". Again, the incorrect semantic understanding of articulated object state can be potentially fixed by using a vision-language model to figure out the mapping between the joint angle and object state. For syntax errors such as undefined variables, one could feed the error back to the LLM and ask it to correct itself. The reward function generation can also be improved to better match the intended goal by incorporating environment feedback into the system [173], which we leave as future work.

Table 7.8: Failure case analysis

Task name	Task description	Failure case
Pack Item In Suitcase	The robot arm places an item .for example, a folded shirt. inside the suitcase	Limited asset functionality: the suitcase cannot be opened.
Open Window Halfway	The robotic arm will open one of the slider translation windows halfway to let fresh air in	Incorrect semantic understanding of articulated object state: setting both joint angles to 0 make the window opened already
Correct Clock Time	The robotic arm corrects the time displayed on the clock based on the standard time	Generated reward refers to undefined variables "standard time"
Wash Hands	The robot arm acts as if its washing hands to demonstrate good hygiene	Reward error in one of the substeps: moving hands back and forth
Fold Chair	The robotic arm will fold the chair to save room or for easy carrying	Wrong reward due to incorrect understanding of the joint state of articulated object. The reward actually encourages unfolding the chair
Unfold Chair	The robotic arm will unfold the folding chair to make it suitable for sitting	Wrong reward due to incorrect understanding of the joint state of articulated object. The reward actually encourages folding the chair
Staple Papers	The robot arm gathers a few loose sheets of paper and uses the stapler to staple them together	Too delicate initial spatial relationship – the task requires the sheet of paper to be initialized into the stapler, which is hard for a random stapler and a sheet of paper sampled from PartNetMobility / Objaverse

Continued on next page

Table 7.8 continued from previous page

Task name	Task description	Failure case
Load Staples into Stapler	The robot arm inserts new staples into the stapler	Asset mismatch: randomly sampled stapler and staple won't easily match each other
Load Paper into Printer	The robot arm loads paper into the printer via the input tray, typically located on the printer body	Limited asset functionality: the printer cannot really be loaded with paper
Print a Document	The robot interacts with the printer to print a document The robot arm first places a document on the printer, then moves the button to initiate the print	Limited asset functionality: the printer cannot really be loaded with paper
Fill Kettle with Water	The robot arm opens the kettle lid, holds a water jug to fill the kettle with water, and then closes the lid	Limited asset functionality: the kettle lid cannot be really moved away from the kettle body
Pour Water from Kettle	The robot arm holds the kettle handle, tilts the kettle to pour water into a cup	Limited asset functionality: the kettle lid cannot be really moved away from the kettle body
Knock On Door	The robotic arm will knock on the door in a typical way a human would	Reward error: not really correct reward function for the knocking motion.
Making Coffee	The robot arm opens the lid of the container, places coffee grounds inside, then closes the lid and starts the brewing process by adjusting the knob	Limited asset functionality: the coffeemachine lid cannot really be moved away from the body
Extract Contents	The robot arm will open the dispenser lid and proceed to extract the contents inside the dispenser	Limited asset functionality: the lid of the dispenser canont be removed from the body to enable the pouring motion.
Fill Dispenser	The robotic arm opens the dispenser lid and then pours the desired content into the dispenser	Limited asset functionality: the lid of the dispenser canont be removed from the body to enable the pouring motion.
Stir Contents in Pot	The robot arm removes the lid of the pot and stirs the pots contents with a stirring spoon	Limited asset functionality: the lid cannot really be removed from the pod
Deliver an Object	The robot arm holds an object, opens the door, passes through, then closes the door behind it This represents the robot arm delivering an object from one room to another	Reward error for delivering an object through the door
Open Eyeglasses	The robotic arm will unfold the legs of the eyeglasses	Incorrect semantic understanding of the object joint state. Setting the joint angle to 0 actually make the eyeglass already unfolded.

Part III

Unified Neural Architectures for Multi-modal Multi-task Policy Learning

Chapter 8

Robotic Policy Learning with 3D Representations and Hierarchical Structure

8.1 Introduction

Solutions to many robot manipulation tasks can be modeled as a sequence of 6-DoF end-effector poses (3D position and orientation). Many recent methods train neural manipulation policies to predict 3D end-effector pose sequences directly from 2D images using supervision from demonstrations [21, 81, 114, 159, 249, 253]. These methods are typically sample inefficient: they often require many trajectories to handle minor scene changes at test time and cannot easily generalize across camera viewpoints and environments, as mentioned in the respective papers and shown in our experiments.

For a robot policy to generalize under translations, rotations, or camera view changes, it needs to be spatially equivariant [339], that is, to map 3D translations and rotations of the input visual scene to similar 3D translations and rotations for the robot's end-effector. Spatial equivariance requires predicting 3D end-effector locations through 2D or 3D action maps, depending on the action space considered, instead of regressing action locations from holistic scene or image features. Trans-

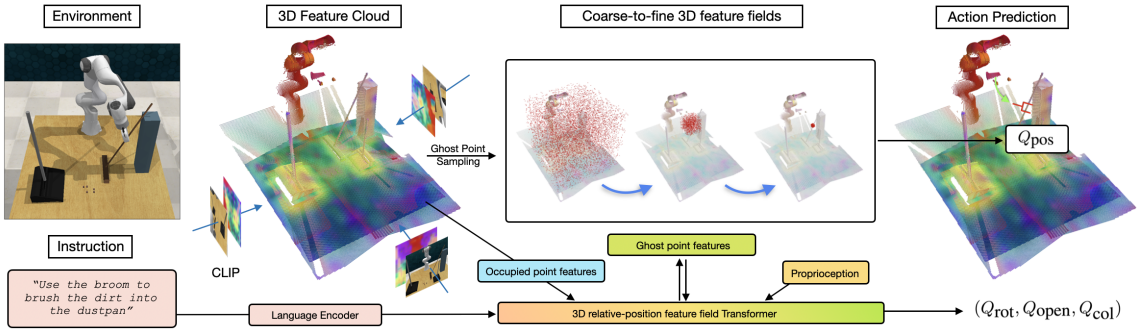


Figure 8.1: **Act3D** is a language-conditioned robot action transformer that learns 3D scene feature fields of arbitrary spatial resolution via recurrent coarse-to-fine 3D point sampling and featurization using relative-position attentions. Act3D featurizes multi-view RGB images with a pre-trained 2D CLIP backbone and lifts them in 3D using sensed depth. It predicts 3D location of the end-effector using classification of the 3D points of the robot's workspace, which preserves spatial equivariance of the scene to action mapping.

porter networks [329] introduced a spatial equivariant architecture for 4-DoF robot manipulation: they re-project RGB-D input images to a top-down image and predict robot end-effector 2D translations through a top-down 2D action map. They showed better generalization with fewer training demonstrations than prior works. However, they are limited to top-down 2D worlds and 4-DoF manipulation tasks. This begs the question: how can we extend spatial equivariance in action prediction to general 6-DoF manipulation?

Developing spatially equivariant 6-DOF manipulation policies requires predicting 3D action maps by classifying 3D points in the robot's workspace as candidates for future 3D locations for the robot's end-effector. Predicting high-resolution 3D action maps, necessary for fine-grained manipulation tasks, poses a computational challenge over their 2D counterparts due to the extra spatial dimension. Voxelizing the robot's 3D workspace and featurizing the 3D voxels at high resolution is computationally demanding [278]. The next end-effector pose might be anywhere in free space, which prevents the use of sparse 3D convolutions [39, 79] to selectively featurize only part of the 3D free space. To address this, recent work of PerAct [249] featurizes 3D voxels using the latent set bottlenecked self-attention operation of Perceiver [111], whose complexity is linear to the number of voxels as opposed to quadratic, as the all-to-all

self attention operations. However, it gives up on spatial disentanglement of features due to the latent set bottleneck. Other methods avoid featurizing points in 3D free space altogether and instead regress an offset for the robot's 3D locations from a detected 2D image contact point [81, 164, 205], which again does not fully comply with spatial equivariance.

In addition to structured 3D representations, we argue for a more hierarhical architecture design for policy networks. While learning manipulation policies from demonstrations is a supervised learning problem, the multimodality and diversity of action trajectories poses significant challenges to machine learning methods. Some tasks, such as placing a cup in a cabinet, can be handled by a policy that provides only a desired goal pose for the cup [247, 249, 327], while others, such as wiping off dirt on the floor, necessitate the policy to generate a continuous action trajectory [177, 279] for the grasped mop. In the current literature, one line of manipulation learning methods models action trajectories from demonstrations, which demonstrated stable training behavior and impressive capability in capturing multimodal action trajectory distributions. Yet, the latter has not yet been tested on long-horizon manipulation tasks. Another line of works casts the problem of robot manipulation as predicting a sequence of discrete end-effector actions on keyframes [76, 78, 239, 327]. This paradigm extracts keyframes from continuous demonstrations and predicts end-effector actions in these keyframes [76, 159, 247, 249]. However, the assumptions behind keyframe prediction hinder its applicability to manipulation tasks that extend beyond pick-and-place type of actions. Therefore, in addition to using 3D representations, we propose to unify these two families of architecture designs.

In this chapter, we introduce Act3D and ChainedDiffuser, language-conditioned transformers for multi-task 6 DoF robot manipulation that predicts continuous 3D trajectories through adaptive 3D spatial computation. Act3D represents the scene as a continuous 3D feature field. It computes a scene-level physical 3D feature cloud by lifting features of 2D foundational models from one or more views using sensed depth. It learns a 3D feature field of arbitrary spatial resolution via recurrent coarse-to-fine 3D point sampling and featurization. At each iteration, the model samples 3D points in the whole workspace and featurizes them using relative spatial cross-attention [262] to the physical 3D feature cloud. Then, we present ChainedDiffuser, a unified structure that combines keypose predicting Act3D and trajectory diffusion.

At a coarse level, ChainedDiffuser predicts macro-step end-effector actions (which we will call *macro-actions*), a high-level task that requires global comprehension of the visual environment and the task to complete, with a global transformer-based action predictor. Then, a low-level trajectory diffuser generates local trajectory segments to connect the predicted macro-actions. In comparison to transformer-based macro-step prediction methods [81, 159, 247, 249], our model predicts smooth trajectories to accommodate tasks that require continuous interactions and collision-free actions. In comparison to diffusion-only trajectory generation methods [36, 117, 224, 279], our hierarchical approach handles long-horizon tasks in a more structured manner and allows different modules to concentrate on the tasks at which they excel.

We test Act3D and ChainedDiffuser in RLBench [113], an established benchmark for learning diverse robot manipulation policies from demonstrations. We set a new state-of-the-art in the benchmark in both single-task and multi-task settings. Specifically, we achieve a 10% absolute improvement over prior SOTA on the single-task setting introduced by HiveFormer [81] with 74 tasks and a 22% absolute improvement over prior SOTA in the multi-task setting introduced by PerAct [249] with 18 tasks and 249 variations. We also validate our approach on a Franka Panda with a multi-task agent trained from scratch on a suite of real-world tasks with a total of just 100 demonstrations (see Figure 8.6). In thorough ablations, we show the importance of the design choices of our architecture, specifically, relative spatial attention, large-scale vision-language pre-trained 2D backbones, high resolution featurization, weight tying across coarse-to-fine attentions, and unification of global keypose prediction and local trajectory diffusion.

8.2 Related Work

Learning robot manipulation from demonstrations Many recent work train multi-task manipulation policies that leverage Transformer architectures [21, 81, 159, 222, 244, 249] to predict robot actions from video input and language instructions. End-to-end image-to-action policy models, such as RT-1 [21], GATO [222], BC-Z [115], and InstructRL [159], directly predict 6-DoF end-effector poses from 2D video and language inputs. They require many thousands of demonstrations to learn spatial reasoning and generalize to new scene arrangements and environments. Transporter

networks [329] and their subsequent variants [78, 239, 248] formulate 4-DoF end-effector pose prediction as pixel classification in 2D overhead images. Thanks to the spatial equivariance of their architecture, their model dramatically increased sample efficiency over previous methods that regress end-effector poses by aggregating global scene features. However, they are limited to top-down 2D planar worlds with simple pick-and-place primitives. 3D policy models of C2F-ARM [114] and PerAct [249] voxelize the robot's workspace and are trained to detect the 3D voxel that contains the next end-effector keypose. Spatially precise 3D pose prediction requires the 3D voxel grid to be high resolution, which comes at a high computational cost. C2F-ARM [114] uses a coarse-to-fine voxelization to handle computational complexity, while PerAct [249] uses Perceiver's latent bottleneck [111] to avoid voxel-to-voxel self-attention operations. Act3D avoids 3D voxelization altogether and instead represents the scene as a continuous resolution 3D feature field. It samples 3D points in the empty workspace and featurizes them using cross-attentions to the physical 3D point features.

Feature pre-training for robot manipulation Many 2D policy architectures bootstrap learning from demonstrations from frozen or finetuned 2D image backbones [115, 193, 206, 318] to increase experience data sample efficiency. Pretrained vision-language backbones can enable generalization to new instructions, objects, and scenes [248, 261]. In contrast, SOTA 3D policy models are typically trained from scratch from colored point clouds input [114, 249, 301]. Act3D uses CLIP pre-trained 2D backbones [217] to featurize 2D image views and lifts the 2D features in 3D using depth [90, 277]. We show that 2D feature pretraining gives a considerable performance boost over training from scratch.

Relative attention layers Relative attentions have shown improved performance in many 2D visual understanding tasks and language tasks [166, 245]. Rotary embeddings [263] implement relative attention efficiently by casting it as an inner-product in an extended position feature space. In 3D, relative attention is imperative as the coordinate system is arbitrary. 3D relative attentions have been used before in 3D Transformer architectures for object detection and point labelling [299, 315]. We show in Section 8.5 that relative attentions significantly boost performance of our

model.

Diffusion Models [97, 197, 254, 255] learn to approximate the data distribution through an iterative denoising process, and have shown impressive results on both unconditional and conditional image generation [52, 220, 228, 231]. In the field of robotics, diffusion models find applications on planning [6, 103, 117], scene re-arrangement [123, 165], controllable motion optimization [119, 335], video generation [57] and imitation learning [36, 224]. Their main advantage is that they can better capture the action trajectory distribution compared to previous generative models. Recent works use diffusion model to predict complete trajectories, often auto-regressively [36, 209]. Instead, we use diffusion models to generate local trajectories that are chained together with macro-actions.

8.3 3D Feature Field Transformers for Multi-Task Robot Manipulation

This section described the detailed architecture of Act3D, as shown in Figure 8.1. It is a policy transformer that, at a given timestep t, predicts a 6-DoF end-effector pose from one or more RGB-D images, a language instruction, and proprioception information regarding the robot's current end-effector pose. Following prior work [81, 112, 159, 249], instead of predicting an end-effector pose at each timestep, we extract a set of *keyposes* that capture bottleneck end-effector poses in a demonstration. A pose is a keypose if (1) the end-effector changes state (something is grasped or released) or (2) velocities approach near zero (a common occurrence when entering pre-grasp poses or entering a new phase of a task). The prediction problem then boils down to predicting the next (best) keypose action given the current observation. At inference time, Act3D iteratively predicts the next best keypose and reaches it with a sampling-based motion planner, following previous works [81, 249].

We assume access to a dataset of n demonstration trajectories. Each demonstration is a sequence of observations $O = \{o_1, o_2, .., o_t\}$ paired with continuous actions $A = \{a_1, a_2, .., a_t\}$ and, optionally, a language instruction l that describes the task. Each observation o_t consists of RGB-D images from one or more camera views; more details are in Section 8.8.2. An action a_t consists of the 3D position and 3D orientation

(represented as a quaternion) of the robot's end-effector, its binary open or closed state, and whether the motion planner needs to avoid collisions to reach the pose:

$$a = \{a_{\text{pos}} \in \mathbb{R}^3, a_{\text{rot}} \in \mathbb{H}, a_{\text{open}} \in \{0, 1\}, a_{\text{col}} \in \{0, 1\}\}$$

Next, we describe the model's architecture in detail.

Visual and language encoder Our visual encoder maps multi-view RGB-D images into a multi-scale 3D scene feature cloud. We use a large-scale pre-trained 2D feature extractor followed by a feature pyramid network [154] to extract multi-scale visual tokens for each camera view. Our input is RGB-D, so each pixel is associated with a depth value. We "lift" the extracted 2D feature vectors to 3D using the pinhole camera equation and the camera intrinsics, based on their average depth. The language encoder featurizes instructions with a large-scale pre-trained language encoder. We use the CLIP ResNet50 [217] visual encoder and language encoders to exploit their common vision-language feature space for interpreting instructions and referential grounding. Our pre-trained visual and language encoders are frozen, not finetuned, during training of Act3D.

Iterative 3D point sampling and featurization Our key idea is to estimate high resolution 3D action maps by learning 3D perceptual representations of free space with arbitrary spatial resolution, via recurrent coarse-to-fine 3D point sampling and featurization. 3D point candidates (which we will call ghost points) are sampled, featurized and scored iteratively through relative cross-attention [262] to the physical 3D scene feature cloud, lifted from 2D feature maps of the input image views. We first sample coarsely across the entire workspace, then finely in the vicinity of the ghost point selected as the focus of attention in the previous iteration, as shown in Figure 8.1. The coarsest ghost points attend to a global coarse scene feature cloud, whereas finer ghost points attend to a local fine scene feature cloud.

Relative 3D cross-attentions We featurize each of the 3D ghost points and a parametric query (used to select via inner-product one of the ghost points as the next best end-effector position in the decoder) independently through cross-attentions to the multi-scale 3D scene feature cloud, language tokens, and proprioception. Featurizing

ghost points independently, without self-attentions to one another, enables sampling more ghost points at inference time to improve performance, as we show in Section 8.5. Our cross-attentions use relative 3D position information and are implemented efficiently with rotary positional embeddings [262]. The absolute locations of our 3D points are never used in our featurization, and attentions only depend on the relative locations of two features.

Decoding actions We score ghost point tokens via inner product with the parametric query to select one as the next best end-effector position a_{pos}. We then regress the end-effector orientation a_{rot} and opening a_{open}, as well as whether the motion planner needs to avoid collisions to reach the pose a_{col}, from the last iteration parametric query with a 2-layer multi-layer perceptron (MLP).

Training Act3D is trained supervised from input-action tuples from a dataset of manipulation demonstrations. These tuples are composed of RGB-D observations, language goals, and keypose actions $\{(o_1, l_1, k_1), (o_2, l_2, k_2), ...\}$. During training, we randomly sample a tuple and supervise Act3D to predict the keypose action k given the observation and goal (o, l). We supervise position prediction a_{pos} at every round of coarse-to-fine with a softmax cross-entropy loss over ghost points, rotation prediction a_{rot} with a MSE loss on the quaternion prediction, and binary end-effector opening a_{open} and whether the planner needs to avoid collisions a_{col} with binary cross-entropy losses.

Implementation details We use three ghost point sampling stages: first uniformly across the entire workspace (roughly 1 meter cube), then uniformly in a 16 centimeter diameter ball, and finally in a 4 centimeter diameter ball. The coarsest ghost points attend to a global coarse scene feature cloud (32x32xn_{cam} coarse visual tokens) whereas finer ghost points attend to a local fine scene feature cloud (the closest 32x32xn_{cam} out of the total 128x128xn_{cam} fine visual tokens). During training, we sample 1000 ghost points in total split equally across the three stages. At inference time, we can trade-off extra prediction precision and task performance for additional compute by sampling more ghost points than the model ever saw at training time ($10,000$ in our experiments). We'll show in ablations in Section 8.5 that our framework is robust to these hyper-parameters but tying weights across sampling

stages and relative 3D cross-attention are both crucial for generalization. We use a batch size 16 on a Nvidia 32GB V100 GPU for 200k steps (one day) for single-task experiments, and a batch size 48 on 8 Nvidia 32GB V100 GPUs for 600K steps (5 days) for language-conditioned multi-task experiments. At test time, we call upon a low-level motion planner to reach predicted keyposes. In simulation, we use native motion planner implementation provided in RLBench, which is a sampling-based BiRRT [133] motion planner powered by Open Motion Planning Library (OMPL) [265] under the hood. For real-world experiments, we use the same BiRRT planner provided by the MoveIt! ROS package [43]. please, see Section 8.8.4 for more details.

8.4 Augmenting Act3D with Local Trajectory Diffusion

This section presents the architecture of ChainedDiffuser, as illustrated in Figure 8.2. ChainedDiffuser combines macro-action prediction with conditional trajectory diffusion. Its input comprises of visual observations of the environment and a natural language description l of the task. At each step, ChainedDiffuser predicts a macro-action $\hat{a}_t$ using a global policy π_{global}, and then feeds $\hat{a}_t$ together with its current end-effector state q_t to a low-level local trajectory generator $\pi_{\text{local}}(q_t, \hat{a}_t)$ to generate dense micro-actions connecting q_t and $\hat{a}_t$, as shown in Figure **??**. Both a_t and q_t share the same space $\mathcal{A} = \{a_{\text{pos}}, a_{\text{rot}}, a_{\text{grip}}\}$, consisting of the end-effector's 3D position a_{pos}, rotation a_{rot} represented as a 4D quaternion, and a binary flag a_{grip} indicating whether the gripper is open. For each task, we assume access to a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, ..., \zeta_m\}$ of m expert demonstrations, where ζ_i contains the language instructions l, visual observations o and end-effector states q_t for all timesteps in the demonstration. The macro-action predictor π_{global} is based on Act3D. Once we obtain the macro-action $\hat{a}_t$ for the current step t, we call upon our diffusion-based local trajectory generator to fill up the gap in-between with micro-actions. We model such trajectory generation as a denoising process [36, 97, 279]: we start with drawing a sequence of S random Gaussian samples $\{\mathbf{x}_s^K\}_{s=1}^S$ in the normalized SE(3) space, and then perform K denoising iterations to transform the noisy trajectories to a sequence of noise-free

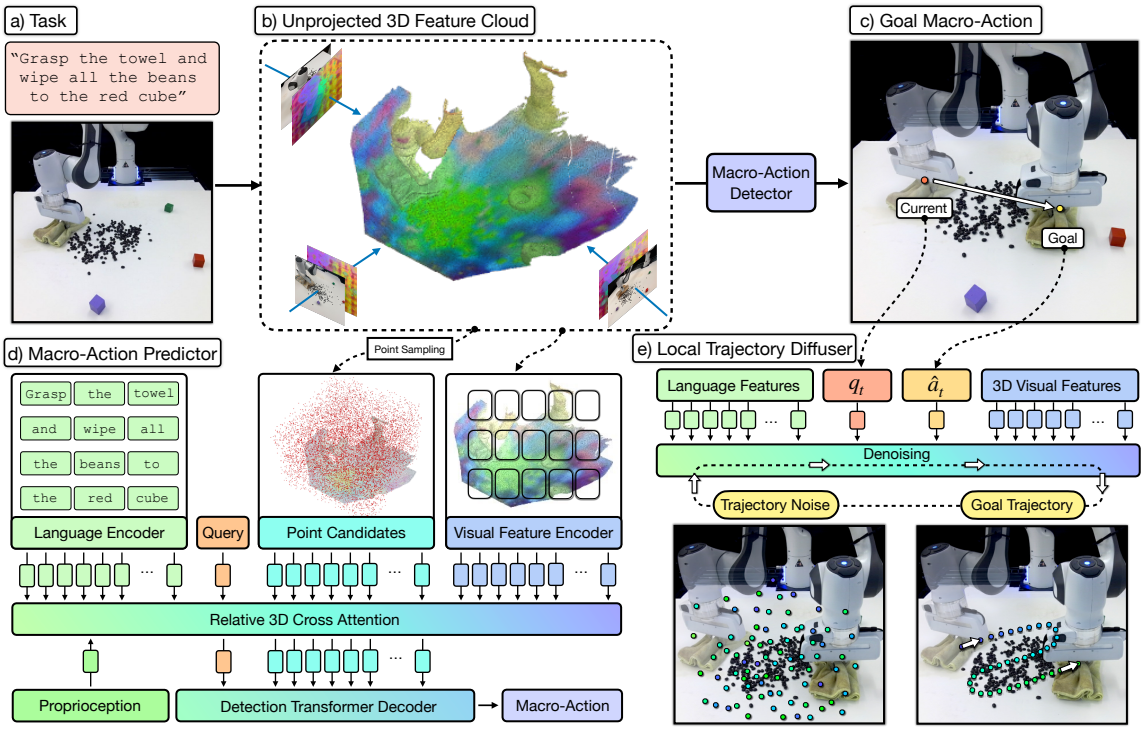


Figure 8.2: **ChainedDiffuser** is a robot manipulation policy architecture that predicts a set of robot keyposes and links them using predicted trajectory segments. It featurizes input multi-view images using pre-trained 2D image backbones and lifts the resulting 2D feature maps to 3D using sensed depth. In (b) we visualize the 3D feature cloud using PCA and keeping the 3 principal components, mapping them to RGB. The model then predicts end-effector keyposes using coarse-to-fine attention operations to estimate a 3D action map for the end-effector's 3D location and regress the robot's 3D orientation, similar to [76] (d). It then links the current end-effector pose to the predicted one with a trajectory predicted using a diffusion model conditioned on the 3D scene feature cloud and predicted keypose (e).

waypoints $\{\mathbf{x}_s^0\}_{s=1}^S$. Each denoising iteration is described by:

$$\mathbf{x}_s^{k-1} = \lambda_k(\mathbf{x}_s^k - \gamma_k \epsilon_\theta(\mathbf{x}_s^k, k)) + \mathcal{N}(0, \sigma_k^2 I), \quad 1 \leq s \leq S \tag{8.1}$$

where ϵ_θ is the noise prediction network, k the denoising step, $\mathcal{N}(0, \sigma_k^2 I)$ the Gaussian noise added at each iteration, and $\lambda_k, \gamma_k, \sigma_k$ are scalar noise schedule functions dependent on k (Section 8.8.5).

The noise prediction network (Figure **??** (e)) is also an attention-based model that

absorbs similar input as the macro-action selector does, i.e., the language instruction l, RGB-D observations o_t and current end-effector state q_t, but additionally conditions on the goal macro-action $\hat{a}_t$ and the denoising timestep k. The language tokens $\mathcal{Z}_{\text{ins}}$, visual tokens $\mathcal{Z}_{\text{vis}}$ and current end-effector state $\mathcal{Z}_{\text{robot}}$ are featurized similarly to the Macro-Action Selector. We use an MLP to encode the goal macro-action into $\mathcal{Z}_{\text{macro}} = \text{MLP}(\hat{a}_t)$. We encode the denoising timestep into $\mathcal{Z}_{\text{time}}$ using sinusoidal positional embeddings [280], and encode the the sampled noise using an MLP into a sequence of tokens $\mathcal{Z}_s^k$. We let this sequence iteratively cross-attend to all encoded inputs first:

$$\tilde{\mathcal{Z}}_s^k = \text{Attn}(\mathcal{Z}_s^k, \langle \mathcal{Z}_{\text{ins}}, \mathcal{Z}_{\text{vis}}, \mathcal{Z}_{\text{robot}}, \mathcal{Z}_{\text{macro}}, \mathcal{Z}_{\text{time}} \rangle),$$

and then self-attend to obtain a finalized $\tilde{\mathcal{Z}}_s^k$ (note that we reuse the same symbol for presentation clarity):

$$\tilde{\mathcal{Z}}_s^k = \text{Attn}(\tilde{\mathcal{Z}}_s^k, \tilde{\mathcal{Z}}_s^k)$$

Again, we use relative positional embeddings to encode all tokens' spatial positions. For the trajectory noise tokens, we additionally encode each sample's temporal position s using sinusoidal positional embeddings. These are added to the respective noise tokens $\mathcal{Z}_s^k$. The contextualized noise sample is then fed into another MLP for noise regression:

$$\epsilon_\theta(\mathbf{x}_s^k, k) = \text{MLP}(\tilde{\mathcal{Z}}_s^k) \tag{8.2}$$

After K denoising steps by substituting Equation 8.2 into 8.1, we convert the denoised samples back to the actual micro-actions by unnormalizing them: $a_{t-1+s} = \text{Unnormalize}(\mathbf{x}_s^0), \quad 1 \leq s \leq S$.

Noise schedulers We model local trajectory optimization as a discrete-time diffusion process, which we implement using the DDPM sampler [97]. DDPM uses a non-parametric time-dependent noise variance scheduler β_k, which defines how much noise is added at each time step. We adopt a scaled linear schedule for the position and a squared cosine schedule for the rotation of each trajectory step.

8.4.1 Implementation and Training Details

ChainedDiffuser takes as input m multi-view RGB-D images of the scene. For experiments in simulation, we use $m = 3$ (*left*, *right*, *wrist*) or $m = 4$ (with an additional *front* view), depending on the settings of the baselines we compare with. For real-world experiments, we use $k = 1$, with a single front-view camera. Each RGB-D image is 256×256 and is encoded to 64×64 visual tokens with CLIP's ResNet50 visual encoder [217]. The demonstration data contains end-effector states for all timesteps. In order to extract macro-actions to supervise the action selection transformer, we use a simple heuristic following previous literature [81, 159, 249]: a timestep is considered to be a keyframe containing macro-action if the gripper opens or closes, or if the robot arm is not moving (when all joint velocities approach zero). All dense actions present in the demonstration are used to supervise the local trajectory diffuser. We resample the dense trajectories between extracted macro-actions to a trajectory of fixed length $S = 50$. We found in practice, denoising fixed number of micro-actions leads to more stable training, and works better than learning variable-length trajectory diffusion with predicted trajectory length. We train both the action detector and the trajectory diffuser jointly, using a cross-entropy (CE) loss to supervise the point candidate selection by predicting a probability distribution q over all point candidates in the pool, and mean-sqaured error (MSE) losses to supervise quaternion, gripper opening and trajectory noise regression:

$$\mathcal{L} = \frac{1}{|\mathcal{D}||\zeta|} \sum_{\zeta \in \mathcal{D}} \sum_{\hat{t} \in \zeta} \left[\text{CE}(q(\{P_i\}^N), q^*(\{P_i\}^N)) + \text{MSE}(\hat{a}_{\hat{t}}, \hat{a}^*_{\hat{t}}) + \sum_{t=\hat{t}}^{\hat{t}+S-1} \text{MSE}(\epsilon_\theta(\mathbf{x}^k_s, k), \epsilon_k) \right], \tag{8.3}$$

where * indicates predicted value, $\hat{a}_{\hat{t}} = \langle \hat{a}_{\text{rot}}, \hat{a}_{\text{grip}} \rangle$, k is a randomly sampled denoising step, and ϵ_k is the sampled ground truth noise. In order to speed up training in practice, we train the first 2 terms till convergence, and then add the 3rd term for joint optimization, as opposed to using ground-truth macro-actions for training the trajectory diffuser. This allows the trajectory diffuser to incorporate certain error recovery capability to handle inaccurate macro-action predictions. In addition, at test time, we normalize the predicted quaternion to ensure it respects the normalization

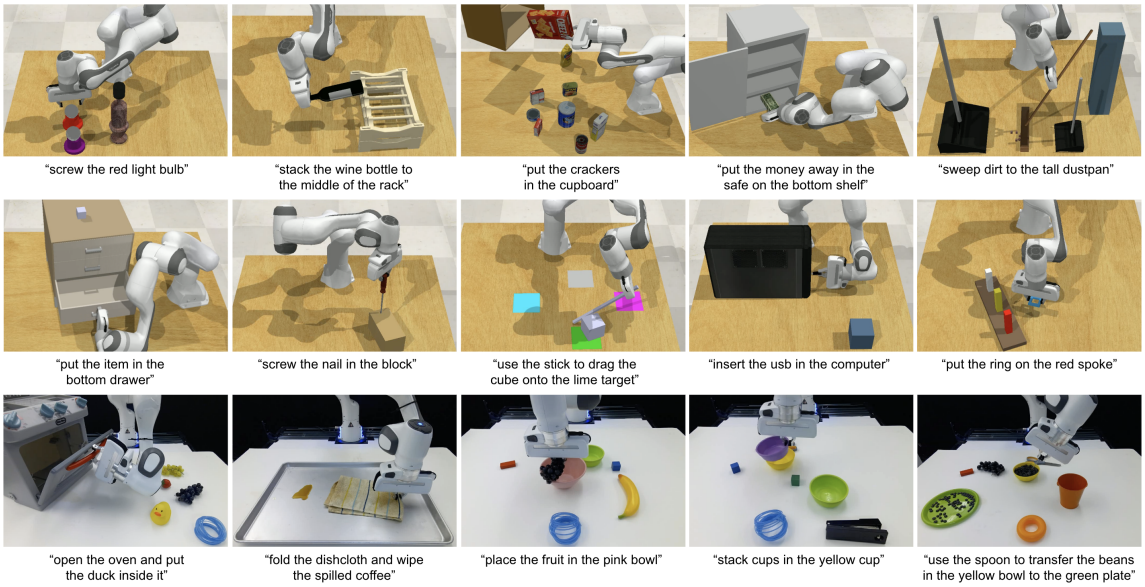


Figure 8.3: **Tasks.** We conduct experiments on 92 simulated tasks in RLBench [113] (only 10 shown), and 8 real-world tasks (only 5 shown).

constraint before feeding it to the robot. We use a batch size of $B = 24$ and AdamW [169] optimizer with a learning rate of $1e-4$ for all our experiments. Our single-task model is trained for 1 day on one A100 GPU, and multi-task model is trained for 5 days on 4 A100 GPUs.

Control Our control algorithm is closed-loop at the macro-action level, which means the macro-action predictor reasons about the surroundings and predicts actions to handle environment changes. At the low-level, our controller is open-loop and follows a Cartesian-space end-effector trajectory composed of the predicted micro-actions using position control, with a control frequency of 10Hz. For our real-robot setup, we use the open source `frankapy` package [331] with ROS, which uses a low-level PID controller at 1kHz.

8.5 Evaluating 3D Representation for Policy Learning

We test Act3D in learning from demonstrations single-task and multi-task manipulation policies in simulation and the real world. We conduct our simulated experiments

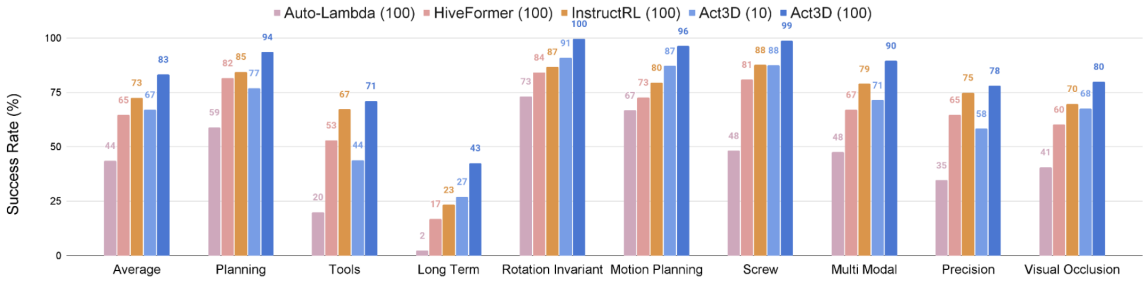


Figure 8.4: **Single-task performance.** On 74 RLBench tasks across 9 categories, Act3D reaches 83% success rate, an absolute improvement of 10% over InstructRL [159], prior SOTA in this setting.

in RLBench [113], an established simulation benchmark for learning manipulation policies, for the sake of reproducibility and benchmarking. Our experiments aim to answer the following questions:

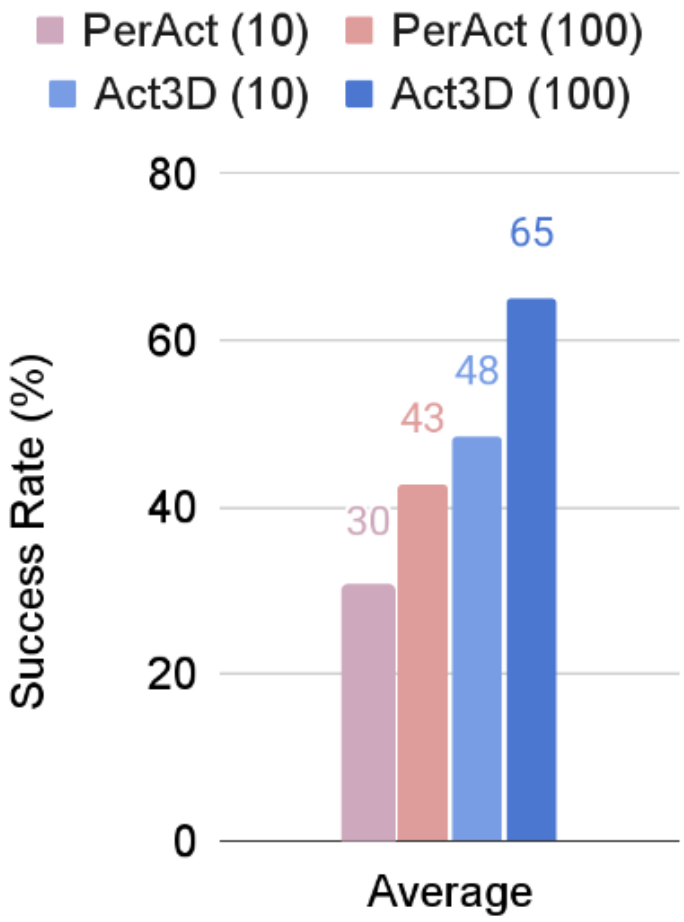
1. How does Act3D compare against SOTA 2D multiview and 3D manipulation policies in single-task and multi-task settings with varying number of training demonstrations?

2. How does Act3D generalize across camera viewpoints compared to prior 2D multiview policies?

3. How do design choices such as relative 3D attention, pre-trained 2D backbones, weight-tied attention layers, and the number of coarse-to-fine sampling stages impact performance?

8.5.1 Evaluation in simulation

Datasets We test Act3D in RLbench in two settings: **1. Single-task** manipulation policy learning. We consider 74 tasks grouped into 9 categories proposed by HiveFormer [81]. Each task includes variations which test generalization to novel arrangements of the same training objects. Each method is trained with 100 demonstrations and evaluated on 500 unseen episodes. **2. Multi-task** manipulation policy learning. We consider 18 tasks with 249 variations proposed by PerAct [249]. Each task includes 2-60 variations, which test generalization to new goal configurations that involve novel object colors, shapes, sizes, and categories. This is a more challenging setting. Each method is trained with 100 demonstrations per task split across



	open drawer		slide block		sweep to dustpan		meat off grill		turn tap		put in drawer		close jar		drag stick		stack blocks	
Method	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
PerAct	68	80	32	72	72	56	**68**	84	**72**	80	16	68	32	60	36	68	**12**	**36**
Act3D	**92**	**93**	**66**	**93**	**82**	**92**	58	**94**	64	**94**	**82**	**90**	**90**	**92**	**52**	**92**	6	12

	screw bulb		put in safe		place wine		put in cupboard		sort shape		push buttons		insert peg		stack cups		place cups	
	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
PerAct	**28**	24	16	44	20	12	0	16	**16**	**20**	56	48	4	0	0	0	0	0
Act3D	26	**47**	**75**	**95**	**32**	**80**	**27**	**51**	7	8	**98**	**99**	**7**	**27**	**8**	**9**	**1**	**3**

Figure 8.5: **Multi-task performance.** On 18 RLBench tasks with 249 variations, Act3D reaches 65% success rate, an absolute improvement of 22% over PerAct [249], prior SOTA in this setting.

variations, and evaluated on 500 unseen episodes per task.

Baselines We compare Act3D with the following state-of-the-art manipulation policy learning methods: **1.** InstructRL [159], a 2D policy that directly predicts 6 DoF poses from image and language conditioning with a pre-trained vision-and-language backbone. **2.** PerAct [249], a 3D policy that voxelizes the workspace and detects the next best voxel action through global self-attention. **3.** HiveFormer [81] and Auto-λ [164], hybrid methods that detect a contact point within an image input, then regress an offset from this contact point. We report numbers from the papers when available.

Evaluation metric We evaluate policies by task completion success rate, the proportion of execution trajectories that lead to goal conditions specified in language instructions.

Single-task and multi-task manipulation results We show single-task quantitative results of our model and baselines in Figure 8.4. Act3D **reaches 83% success rate, an absolute improvement of 10% over InstructRL [159], prior SOTA in this setting**, and consistently outperforms it across all 9 categories of tasks. With only 10 demonstrations per task, Act3D is competitive with prior SOTA using 100 demonstrations per task. Act3D outperforms 2D methods of InstructRL and Hiveformer because it reasons directly in 3D. For the same reason, it generalizes much better than them to novel camera placements, as we show in Table 8.5.

We show multi-task quantitative results of our model and PerAct in Figure 8.5. Act3D reaches 65% success rate, an absolute improvement of 22% over PerAct, prior SOTA in this setting, consistently outperforming it across most tasks. **With only 10 demonstrations per task, Act3D outperforms PerAct using 100 demonstrations per task.** Note that Act3D also uses less than a third of PerAct's training computation budget: PerAct was trained for 16 days on 8 Nvidia V100 GPUs while we train for 5 days on the same hardware. Act3D outperforms PerAct because its coarse-to-fine relative attention based 3D featurization of the 3D workspace is more effective than the perceiver's latent bottleneck attention in generating spatially disentangled features.

Table 8.1: Ablations for different modules of Act3D.

		Average success rate in single-task setting (5 tasks)
Core design choices	Full Act3D	**98.1**
	Only 2 stages of coarse-to-fine sampling	93.6
	No weight tying across stages	80.6
	Absolute 3D positional embeddings	55.4
	Attention to only global coarse visual features	89.8
	Only 1000 ghost points at inference time	93.2
Viewpoint changes	Act3D	**74.2**
	HiveFormer	20.4
		Multi-task setting (18 tasks)
Backbone	CLIP ResNet50 backbone	**65.1**
	ImageNet ResNet50 backbone	53.4

8.5.2 Ablations

We ablate the impact of our design choices in Table 8.5. We perform most ablations in the single-task setting on 5 tasks: pick cup, put knife on chopping board, put money in safe, slide block to target, take umbrella out of stand. We ablate the choice of pre-trained 2D backbone in the multi-task setting with all 18 tasks.

Generalization across camera viewpoints: We vary camera viewpoints at test time for both Act3D and HiveFormer [81]. The success rate drops to 20.4% for HiveFormer, a relative 77% drop, while Act3D achieves 74.2% success rate, a 24% relative drop. This shows detecting actions in 3D makes Act3D more robust to camera viewpoint changes than multiview 2D methods that regress offsets.

Weight-tying and coarse-to-fine sampling: All 3 stages of coarse-to-fine sampling are necessary: a model with only 2 stages of sampling and regressing an offset from the position selected at the second stage suffers a 4.5% performance drop. Tying weights across stages and relative 3D positional embeddings are both crucial; we observed severe overfitting without, reflected in respective 17.5% and 42.7% performance drops. Fine ghost point sampling stages should attend to local fine visual features with precise positions: all stages attending to global coarse features leads to a 8.3% performance drop. Act3D can effectively trade off inference computation for performance: sampling 10,000 ghost points, instead of the 1,000 the model was

Table 8.2: Success rate for 8 real-world tasks.

Task	# Train	Success
reach target	10	10/10
duck in oven	15	6/10
wipe coffee	15	7/10
fruits in bowl	10	8/10
stack cups	15	6/10
transfer beans	15	5/10
press handsan	10	10/10
uncrew cap	10	8/10

trained with, boosts performance by 4.9%.

Pre-training 2D features: We investigate the effect of the pre-trained 2D backbone in the multi-task setting where language instructions are most needed. A ResNet50 [217] backbone pre-trained with CLIP improves success rate by 8.7% over a ResNet50 backbone pre-trained on ImageNet.

We found Random crops of RGB-D images to boost performance but yaw rotation perturbations did not help. The model is robust to variations in hyperparameters such as the diameter of ghost point sampling balls or the number of points sampled during training. For additional ablations regarding augmentations and sensitivity to hyperparameters, please see section 8.8.7.

8.5.3 Evaluation in real-world

In our real-world setup, we conduct experiments with a Franka Emika Panda robot and a single Azure Kinect RGB-D sensor. We consider 8 tasks (Figure 8.6) that involve interactions with multiple types of objects, spanning liquid, articulated objects, and deformable objects. For each task, we collected 10 to 15 kinesthetic demonstrations and trained a languaged-conditioned multi-task model with all of them. We report the success rate on 10 episodes per task in Table 8.2. Act3D can capture semantic knowledge in demonstration well and performs reasonably well on all tasks, even with a single camera input. One major failure case comes from noisy depth sensing: when the depth image is not accurate, the selected point results in imprecise action

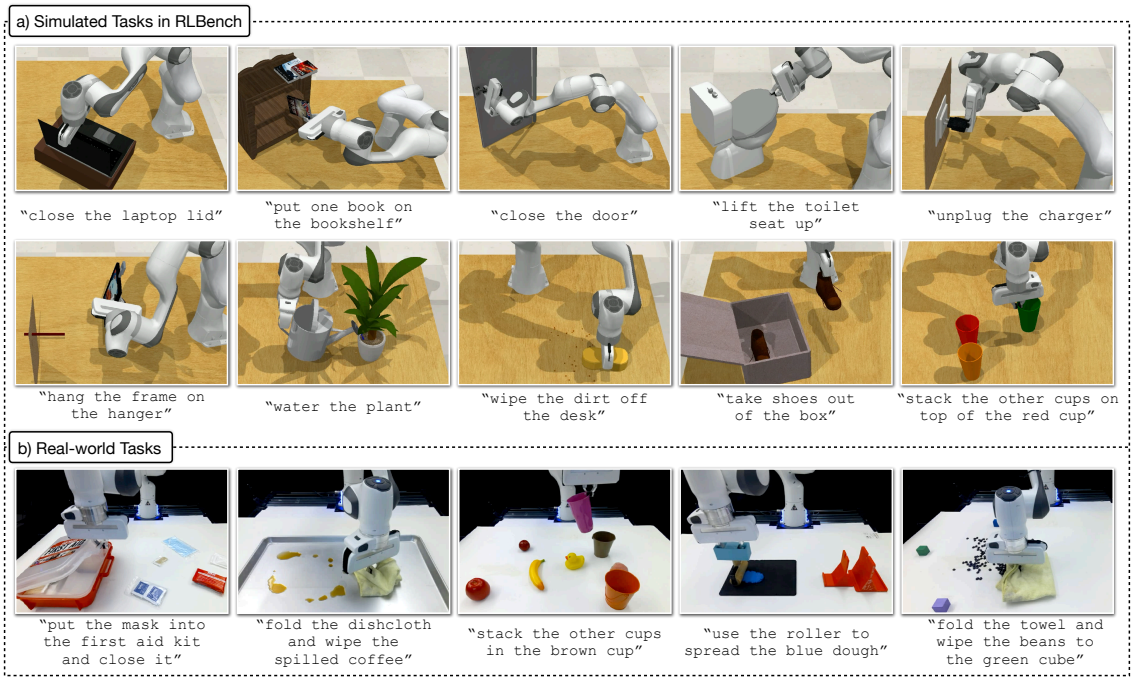


Figure 8.6: Simulation and real-world tasks we evaluate on.

prediction. Leveraging multi-view input for error correction could improve this, and we leave this for future work. For videos of the robot executing the tasks, please see our project website.

8.6 Evaluating the Performance with Additional Local Trajectory Diffusion

In this section, we test ChainedDiffuser, which augments Act3D with local trajectory diffusion, in various manipulation tasks. Our experiments aim to answer the following questions:

- How does ChainedDiffuser compare to previous SOTA 2D and 3D manipulation methods?
- Is macro-action prediction helpful in guiding trajectory generation?
- Does ChainedDiffuser work in the real-world where only a single camera and limited number of demonstrations are available?

Table 8.3: Success rates in 10 single-tasks of the Hiveformer experimental setting.

	pick & lift	pick-up cup	push button	put knife	put money	reach target	slide block	stack wine	take money	take umbrella	Mean
Auto-λ [164]	87	78	95	31	62	100	36	23	38	37	55.0
HiveFormer [81]	92	77	100	70	96	100	95	82	82	90	88.4
InstructRL [159]	98	85	**100**	85	**99**	100	**98**	**93**	90	93	93.8
ChainedDiffuser (ours)	**98**	**94**	96	**91**	98	**100**	95	90	**100**	**96**	**95.8**

8.6.1 Experimental Setting

Again, we run experiments in simulation using RLBench [113]. We report success rates in each task averaged over 100 unseen test episodes. For baselines, when possible, we use the official numbers reported in their papers.

Baselines We compare ChainedDiffuser with the following baselines:

1. Auto-λ [164] and HiveFormer [81], policy learners that operate on multi-view 2.5D images and predict actions by offseting detected points in the input images.
2. *InstructRL* [159], a policy that operates on multi-view 2D images with pre-trained vision and language encoders, and directly predicts 6-DoF end-effector actions.
3. *Act3D* [76], a policy that predicts keyframe end-effector macro-actions with a 3D action detection transformer and relies on low-level motion planner to connect macro-actions.
4. *Open-loop trajectory diffusion*, which is ChainedDiffuser without the macro-action detector, making it a trajectory diffusion model.
5. *Act3D+ trajectory regression*, which replaces the local trajectory diffuser in ChainedDiffuser with a deterministic trajectory regression

Dataset We consider the following single-task experimental settings:

- 10 tasks considered in the Auto-λ [164] experimental setup. These tasks are considered by many prior works and this allows us to compare our performance with them.
- 10 tasks in RLbench we identify to require continuous interaction with the environment, such as `wipe_desk` where a wiping trajectory is needed to remove

Table 8.4: Success rates on challenging tasks for motion planners.

	unplug charger	close door	open box	open fridge	frame off hanger	open oven	books on shelf	wipe desk	cup in cabinet	shoes out of box	Mean
Act3D [76]	48	9	9	19	66	2	34	4	0	19	21.0
Open-loop trajectory diffusion	65	21	46	37	43	16	40	34	6	9	31.7
Act3D + trajectory regression	95	5	95	60	77	17	68	**70**	40	67	59.6
ChainedDiffuser (ours)	**95**	**76**	**96**	**68**	**85**	**86**	**92**	65	**68**	**78**	**80.9**

the dirt from a desk, and `open_fridge` where a local trajectory needs to adhere to the kinematic constraint when the robot is grasping the door handle. Most tasks in RLBench can be reasonably solved with only macro-action prediction and motion planners. This set of tasks we consider highlights the limitation of these approaches.

Results We train single-task ChainedDiffuser and the baselines. For *Auto-λ*, *HiveFormer* and *InstructRL* we use the numbers reported in the corresponding papers. We show quantitative results in Tables 8.3 and 8.4. ChainedDiffuser consistently achieves better performance than prior methods on all task categories. On the set of challenging tasks for motion planners, ChainedDiffuser gives a significant boost of 60% on average. ChainedDiffuser improves upon open-loop trajectory diffusion model, which demonstrates that delegating global macro-action prediction to a high-level policy to guide local trajectory diffusion helps. *Act3D+ trajectory regression* struggles where multi-modal trajectories are present in demonstrations, e.g. `cup_in_cabinet` where multimodal trajectories exist for grasping the cup and feeding into the cabinet in the training set. This demonstrates that modeling trajectory generation as a multi-step denoising process is advantageous over regression-based model, which aligns with conclusions from previous literature [36].

8.6.2 Limitations

Our method currently has the following limitations: **1)** Our trajectory diffuser is conditioned on end-effector poses in SE(3) space. It would be ideal to extend it to full joint configuration space for more flexible trajectory prediction. **2)** Our model performs closed-loop control on the macro-action level, which restricts its flexibility in highly dynamic environments. That said, our framework can be easily extended with

closed-loop re-planning at the micro-action level, making the policy more robust to environment dynamics, which we leave as our future work. **3)** Following the standard setting in RLBench, our method assumes access to calibrated cameras. We believe this assumption is valid as mobile robots performing household tasks for humans in the future should have cameras attached to the robots, where these cameras can be calibrated when coming out of the factories.

8.7 Conclusion

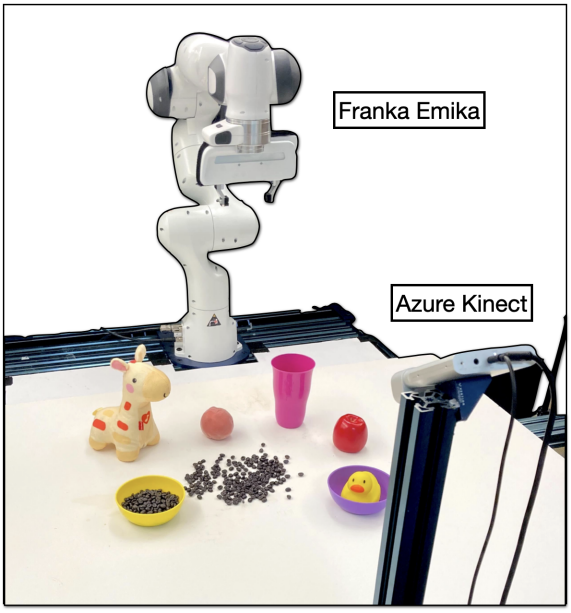
In this chapter, we first presented Act3D, a language-conditioned policy transformer that predicts continuous resolution 3D action maps for multi-task robot manipulation. Act3D represents the scene using a continuous resolution 3D feature map, obtained by coarse-to-fine 3D point sampling and attention-based featurization. We then attempted to unify it with local trajectory diffusion and presented ChainedDiffuser. They together sets a new state-of-the-art in RLBench, an established robot manipulation benchmark, and solves diverse manipulation tasks in the real world from a single RGB-D camera view and a handful of demonstrations. Our ablations quantified the contribution of relative 3D attentions, 2D feature pre-training, and weight tying during coarse-to-fine iterations. We also demonstrated that by unifying both transformer-based macro-action detection and diffusion-based trajectory generation, our method achieves the best of both families and addresses their respective limitations.

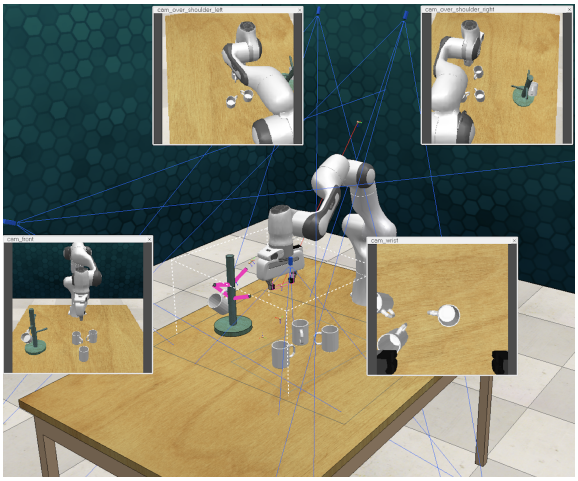
8.8 Additional Details

8.8.1 Real-world Setup

Our real-robot setup contains a Franka Panda robotic arm equipped with a parallel jaw gripper. We get RGB-D input from a single Azure Kinect sensor at a front view at 30Hz. The image input is of resolution 1280×720, we crop and downsample it to 256×256. We calibrate the extrinsics of the camera with respect to the robot base using the `easy_handeye`[1] ROS package. We extract keyposes from demonstrations

[1] https://github.com/IFL-CAMP/easy_handeye





in the same was as in simulation. Our real-world multi-task policy is trained on 4 V100 GPUs for 3 days, and we run inference on a desktop with a single RTX4090 GPU. For robot control, we use the open-source `frankapy`[2] package to send real-time position-control commands to the robot.

[2] https://github.com/iamlab-cmu/frankapy

8.8.2 RLBench Simulation Setup

To ensure fair comparison with prior work, we use $n_{\text{cam}} \in \{3, 4\}$ cameras for simulated experiments depending on the evaluation setting. In our single-task evaluation setting first proposed by HiveFormer [81], we use the same 3 cameras they do $\{O_{\text{left}}, O_{\text{right}}, O_{\text{wrist}}\}$. In our multi-task evaluation setting first proposed by PerAct [249], we use the same 4 cameras they do $\{O_{\text{front}}, O_{\text{left}}, O_{\text{right}}, O_{\text{wrist}}\}$.

8.8.3 RLBench Tasks

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

Figure 8.7: **PerAct [249] tasks.** We adopt the multi-task multi-variation setting from PerAct [249] with 18 tasks and 249 unique variations across object placement, color, size, category, count, and shape.

We adapt the single-task setting of HiveFormer [81] with 74 tasks grouped into 9 categories according to their key challenges. The 9 task groups are defined as follows:

- The **Planning** group contains tasks with multiple sub-goals (e.g. picking a basket ball and then throwing the ball). The included tasks are: basketball in hoop, put rubbish in bin, meat off grill, meat on grill, change channel, tv on, tower3, push buttons, stack wine.
- The **Tools** group is a special case of planning where a robot must grasp an object to interact with the target object. The included tasks are: slide block to target, reach and drag, take frame off hanger, water plants, hang frame on hanger, scoop with spatula, place hanger on rack, move hanger, sweep to dustpan, take plate off colored dish rack, screw nail.
- The **Long term** group requires more than 10 macro-steps to be completed. The included tasks are: wipe desk, stack blocks, take shoes out of box, slide cabinet open and place cups.
- The **Rotation-invariant** group can be solved without changes in the gripper rotation. The included tasks are: reach target, push button, lamp on, lamp off,

push buttons, pick and lift, take lid off saucepan.

- The **Motion planner** group requires precise grasping. As observed in [81] such tasks often fail due to the motion planner. The included tasks are: toilet seat down, close laptop lid, open box, open drawer, close drawer, close box, phone on base, toilet seat up, put books on bookshelf.
- The **Multimodal** group can have multiple possible trajectories to solve a task due to a large affordance area of the target object (e.g. the edge of a cup). The included tasks are: pick up cup, turn tap, lift numbered block, beat the buzz, stack cups.
- The **Precision** group involves precise object manipulation. The included tasks are: take usb out of computer, play jenga, insert onto square peg, take umbrella out of umbrella stand, insert usb in computer, straighten rope, pick and lift small, put knife on chopping board, place shape in shape sorter, take toilet roll off stand, put umbrella in umbrella stand, setup checkers.
- The **Screw** group requires screwing an object. The included tasks are: turn oven on, change clock, open window, open wine bottle.
- The **Visual Occlusion** group involves tasks with large objects and thus there are occlusions from certain views. The included tasks are: close microwave, close fridge, close grill, open grill, unplug charger, press switch, take money out safe, open microwave, put money in safe, open door, close door, open fridge, open oven, plug charger in power supply

8.8.4 Further Architecture Details

Relative 3D cross-attentions We featurize each of the 3D ghost points and a parametric query (used to select via inner-product one of the ghost points as the next best end-effector position in the decoder) independently through cross-attentions to the multi-scale 3D scene feature cloud, language tokens, and proprioception. Featurizing ghost points independently, without self-attentions to one another, enables sampling more ghost points at inference time to improve performance, as we show in Section 8.5. Our cross-attentions use relative 3D position information and are implemented efficiently with rotary positional embeddings [262].

Given a point $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ and its feature $\mathbf{x} \in \mathbb{R}^d$, the rotary position encoding function **PE** is defined as:

$$\mathbf{PE}(\mathbf{p}, \mathbf{x}) = \mathbf{M}(\mathbf{p})\mathbf{x} = \begin{bmatrix} \mathbf{M}_1 & & \\ & \ddots & \\ & & \mathbf{M}_{d/6} \end{bmatrix} \mathbf{x} \tag{8.4}$$

$$\mathbf{M}_k = \begin{bmatrix} \cos x\theta_k & -\sin x\theta_k & 0 & 0 & 0 & 0 \\ \sin x\theta_k & \cos x\theta_k & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos y\theta_k & -\sin y\theta_k & 0 & 0 \\ 0 & 0 & \sin y\theta_k & \cos y\theta_k & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos z\theta_k & -\sin z\theta_k \\ 0 & 0 & 0 & 0 & \sin z\theta_k & \cos z\theta_k \end{bmatrix} \tag{8.5}$$

where $\theta_k = \frac{1}{10000^{6(k-1)/d}}$, $k \in \{1, .., d/6\}$. The dot product of two positionally encoded features is

$$\mathbf{PE}(\mathbf{p}_i, \mathbf{x}_i)^T \mathbf{PE}(\mathbf{p}_j, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{M}(\mathbf{p}_i)^T \mathbf{M}(\mathbf{p}_j)\mathbf{x}_j = \mathbf{x}_i^T \mathbf{M}(\mathbf{p}_j - \mathbf{p}_i)\mathbf{x}_j \tag{8.6}$$

which depends only on the relative positions of points $\mathbf{p}_i$ and $\mathbf{p}_j$.

We extract two feature maps per 256x256 input image view: 32x32 coarse visual tokens and 128x128 fine visual tokens. We use three ghost point sampling stages: first uniformly across the entire workspace (roughly 1 meter cube), then uniformly in a 16 centimeter diameter ball, and finally in a 4 centimeter diameter ball. The coarsest ghost points attend to a global coarse scene feature cloud (32x32xn_{cam} coarse visual tokens) whereas finer ghost points attend to a local fine scene feature cloud (the closest 32x32xn_{cam} out of the total 128x128xn_{cam} fine visual tokens). During training, we sample 1000 ghost points in total split equally across the three stages. At inference time, we can trade-off extra prediction precision and task performance for additional compute by sampling more ghost points than the model ever saw at training time (10,000 in our experiments). We'll show in ablations in Section 8.5 that our framework is robust to these hyper-parameters but tying weights across sampling

stages and relative 3D cross-attention are both crucial for generalization. We use 2 layers of cross-attention and an embedding size 60 for single-task experiments and 120 for multi-task experiments. Training samples are augmented with random crops of RGB-D images and ±45.0 yaw rotation perturbations (only in the real world as this degrades performance in simulation as we'll show in Section 8.5). The cropping operation is performed on aligned RGB and depth frames together, thus maintain pixel-level correspondence. We use a batch size 16 on a Nvidia 32GB V100 GPU for 200k steps (one day) for single-task experiments, and a batch size 48 on 8 Nvidia 32GB V100 GPUs for 600K steps (5 days) for language-conditioned multi-task experiments. At test time, we call a low-level motion planner to reach predicted keyposes. In simulation, we use native motion planner implementation provided in RLBench, which is a sampling-based BiRRT [133] motion planner powered by Open Motion Planning Library (OMPL) [265] under the hood. For real-world experiments, we use the same BiRRT planner provided by the MoveIt! ROS package [43].

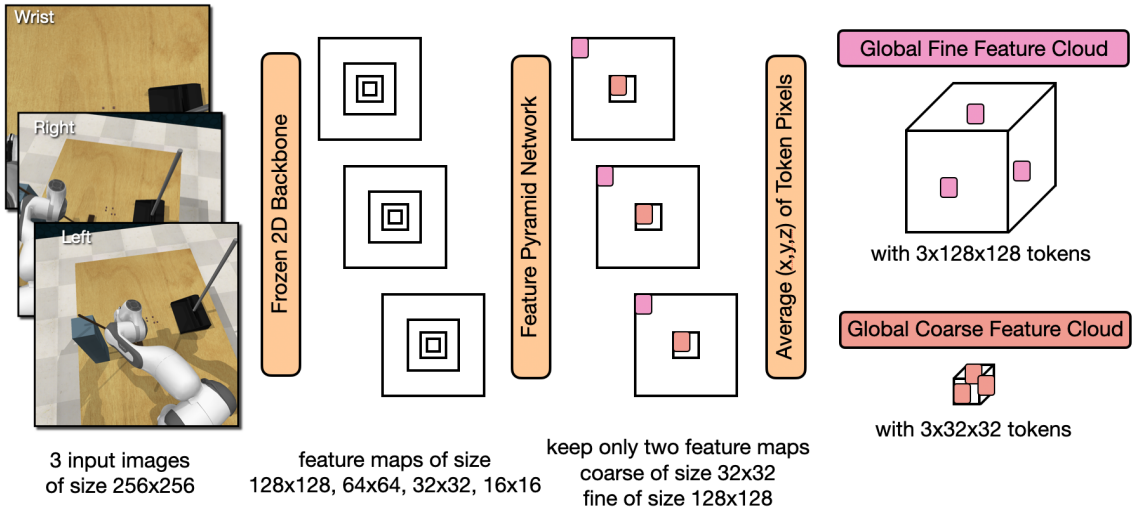


Figure 8.8: **Scene Feature Cloud Generation**. We encode each image independently with a pre-trained and frozen vision backbone to get multi-scale feature maps, pass these feature maps through a feature pyramid network and retain only two: a coarse feature map (at a granularity that lets ghost points attend to all tokens within GPU memory) and a fine feature map (as spatially precise as afforded by input images and the backbone). We lift visual tokens from these two feature maps for each image to 3D scene feature clouds by averaging the positions of pixels in each 2D visual token.

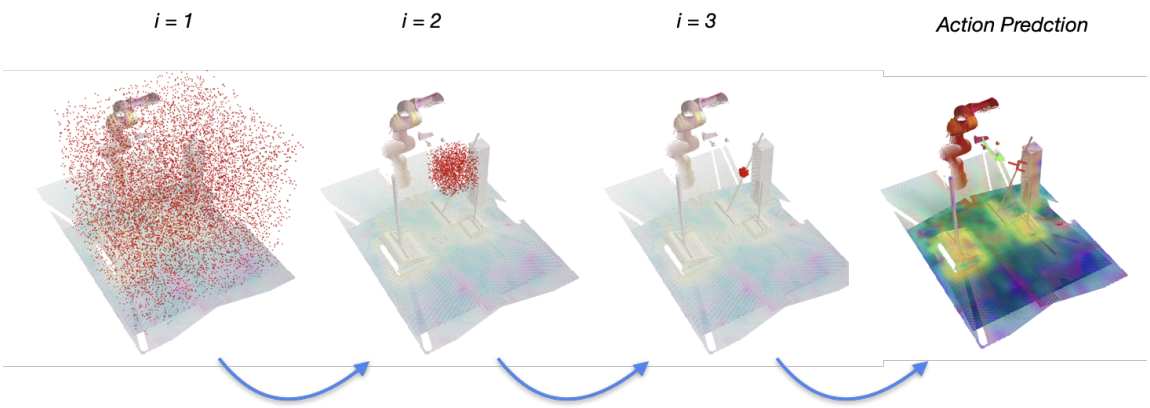


Figure 8.9: **Iterative Ghost Point Sampling, Featurization, and Selection**.

8.8.5 Noise schedulers for Local Trajectory Diffuser

We model local trajectory optimization as a discrete-time diffusion process, which we implement using the DDPM sampler [97]. DDPM uses a non-parametric time-dependent noise variance scheduler β_k, which defines how much noise is added at each time step. We adopt a scaled linear schedule for the position and a squared cosine schedule for the rotation of each trajectory step.

$$x_{k-1} = \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}(x_k - \epsilon_\theta(x_k, k, \mathbf{c})) + \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}x_k + \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k\mathbf{z} \tag{8.7}$$

By defining $\alpha_k = 1 - \beta_k$, and $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$, we can now obtain the analytical form of $\lambda_k, \gamma_k, \sigma_k$ in Equation 8.1 as follows:

$$\lambda_k = \frac{1}{\sqrt{\alpha_k}} \tag{8.8}$$

$$\gamma_k = \frac{1-\alpha_k}{\sqrt{1-\bar{\alpha}_k}} \tag{8.9}$$

$$\sigma_k = \frac{1-\bar{\alpha}_{k+1}}{1-\bar{\alpha}_k}\beta_k \tag{8.10}$$

where k is the diffusion denoising timestep.

8.8.6 High Precision Experiments

In this section, we further investigate the ability of Act3D to improve over existing 3D methods that voxelize the workspace for high-precision tasks. We compare two variants of Act3D against PerAct [249] on three high-precision tasks in success rate. The first Act3D variant is the standard architecture used in the remainder of our experiments operating on 256x256 input image views; the second operates on higher resolution 512x512 input image views, from which it extracts four times as many visual tokens with more precise 3D positions. This further tests the ability of Act3D to provide high precision by processing higher-resolution RGB-D views at the cost of extra compute.

Method	insert peg	sort shape	screw nail
PerAct	16	31	12
Act3D (256x256)	29	34	31
Act3D (512x512)	**47**	**43**	**55**

Act3D improves over PerAct on high precision tasks and can further benefit from higher resolution RGB-D images, at the cost of extra compute.

8.8.7 Further ablations

Augmentations: Random crops of RGB-D images boost success rate by 6.5%, but yaw rotation perturbations drop it by 11.9%. This is in line with PerAct [249] results in RLBench.

Hyperparameter sensitivity: Act3D is robust to variations in hyperparameters. Doubling the diameter of ghost point sampling balls from (16 cm, 4 cm) to (32 cm, 8 cm) drops success rate by 1.5% and halving it to (8 cm, 2 cm) by 6.9%. Halving the total number of ghost points sampled from 1,000 to 500 drops success rate by 2.3% whereas doubling it to 2,000 increases success rate by 0.3%. We use 1,000 ghost points in our experiments to allow training with a single GPU per task.

Table 8.5: Ablations for different modules in ChainedDiffuser

	Model	Average success rate in single-task setting (5 tasks)
Core design choices	Best Act3D model (evaluated in Fig. 8.4)	**98.1**
	Only 2 stages of coarse-to-fine sampling: full workspace, 16 cm ball, regress an offset	93.6
	No weight tying across stages	80.6
	Absolute 3D positional embeddings	55.4
	Attention to only global coarse visual features	89.8
	Only 1000 ghost points at inference time	93.2
Viewpoint changes	Best Act3D model (evaluated in Fig. 8.4)	**74.2**
	HiveFormer	20.4
Augmentations	No image augmentations	**91.6**
	With rotation augmentations	86.2
Hyperparameter sensitivity	Double sampling ball diameters: 32 cm and 8 cm	96.6
	Halve sampling ball diameters: 8 cm and 2 cm	91.2
	500 ghost points at training time	95.8
	2000 ghost points at training time (need 2 GPUs)	**98.4**
		Multi-task setting (18 tasks)
Backbone	CLIP ResNet50 backbone	**65.1**
	ImageNet ResNet50 backbone	53.4
	No backbone (raw RGB)	45.2

Chapter 9

Conclusion and Future Work

In this thesis, we have presented a complete framework that could potentially leads to general purpose robots. First of all, we presented both learning-based and rule-based powerful simulators that can leverage compute to create physical worlds for robots to explore and interact with. This thread of work includes 3D-OES and HyperDynamics, two neural dynamics models that explore 3D representation and hierarchical network structures for better modeling the dynamics of the world in a data driven manner. 3D-OES proposes to use structured 3D representation and geometry-aware neural backbones to simulate the physical world. HyperDynamics than proposes to use HyperNetworks as a backbone to hierarchically infer the physical world properties before performing dynamics prediction. After these, we visit the driving force behind learning dynamics model and robotic policy – rule-based robotics simulators. In FluidLab, We extend the capability of existing simulators to support comprehensive simulation capabilities for modeling deformable objects, and added support for differentiability, while demonstrating the usefulness of gradient information in learning contact-rich and fine-grained manipulation tasks. Lastly, we briefly presented Genesis, a large-scale collaboration between multiple universities and industry research labs, aiming for building a next-generation simulation infrastructure for robotics research. Genesis integrates multiple state-of-the-art physics solvers into a unified simulation framework and delivers an unprecedented simulation speed for massively parallel training. In addition, Genesis is designed in a fully objected-oriented structure and presents a user friendly interface, for lowering the barrier of using robotic simulators.

In the second part, we presented a novel paradigm termed *generative simulation*, which advocates for extracting common sense and generative capability from existing multi-modal foundation models to generate semantic and static knowledge for automated robotic skill training. Specifically, we first discussed Gen2Sim, an initial proof of concept study focused on table top manipulation settings involving rigid and articulated bodies. We then presented RoboGen, a more comprehensive framework that extends to more complex scene generation and manipulation task settings. We demonstrate that our system can generate and auto-solve temporally extended tasks, and the diversity of our generated environments and skills outperform established human-crafted benchmarking datasets.

In the third part, we explore how to use 3D representations and hierarchical inductive bias to build unified policy learning architectures. We present Act3D, a multi-task policy architecture that builds upon explicit 3D internal representations, which demonstrates state-of-the-art imitation learning performance. We then propose to merge such keypose-based architectures with diffusion-based trajectory modeling methods into a unified architecture, which we demonstrate to excel in tasks that were challenging previously.

Notably, we also started the Genesis effort during the course of conducting this line of work. robotics researchers have long been limited by usability issues and opaqueness of available simulators. Genesis aims to provide a fully transparent, user-friendly ecosystem where contributors from both physics simulation and robotics backgrounds can come together to collaboratively create a high-efficiency, realistic (both physically and visually) virtual world for robotics research and beyond. In addition, we recognize the wealth of innovative algorithms that are continuously developed by the computer graphics community for both simulation and rendering, and Genesis aims to create a long-term collaborative effort to bring together these algorithms to create a realistic and compute-powered virtual realm for embodied and physical intelligence to emerge. Genesis will serve as not only a simulation infrastructure, but also aims to be a generative data engine that natively supports generative simulation. In our future work, we aim to keep polishing this data engine platform, and bring in data generation support for various modalities of data: not only assets, indoor scenes and robotic policy data, but also other generative modules including character motion, camera motion, large scale terrain and environments,

robot morphologies and configurations. The ultimate goal of Genesis is to build a fundamental data source, where we can extract various spatially-consistent, visually realistic, and physical accurate data. We believe this will make a significant impact on not only the robotics community, but also other relevant data-driven research fields.

To conclude, this thesis addresses three foundational pillars for building data-driven intelligent robotic systems: creating robust simulation infrastructure for data collection, developing automated pipelines to autonomously generate high-quality data in a self-supervised manner, and designing policy architectures that excel in imitating collected demonstration data. These efforts represent an initial step toward the ambitious goal of autonomously generating high-quality robotic demonstration data and effectively learning from them. We envision a future where data-driven robot learning is significantly empowered by advanced computational infrastructures and data-generation pipelines. Our ultimate aspiration is to faithfully recreate the complexities of the physical world in highly accurate and controllable simulated realms, and we remain committed to advancing this vision.

Bibliography

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL http://arxiv.org/abs/1606.07419. 2.2

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pages 5074–5082, 2016. 3.4.1

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 7.2

[4] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauzá, Leslie Pack Kaelbling, Joshua B. Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. *CoRR*, abs/1808.03246, 2018. URL http://arxiv.org/abs/1808.03246. 2.2

[5] Anurag Ajay, Maria Bauza, Jiajun Wu, Nima Fazeli, Joshua Tenenbaum, Alberto Rodriguez, and Leslie Kaelbling. Combining physical simulators and object-based networks for control, 04 2019. 3.2.1

[6] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, T. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *ArXiv*, abs/2211.15657, 2022. 8.2

[7] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 7.2

[8] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 2.2, 2.4, 1, 3.4.1

[9] Rika Antonova, Peiyang Shi, Hang Yin, Zehang Weng, and Danica Kragic Jensfelt. Dynamic environments with deformable objects. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. 4.3.4

[10] Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. *arXiv preprint arXiv:2207.00167*, 2022. 4.1, 4.2

[11] Ershad Banijamali, Rui Shu, Hung Bui, Ali Ghodsi, et al. Robust locally-linear controllable embedding. In *International Conference on Artificial Intelligence and Statistics*, pages 1751–1759, 2018. 3.1, 3.2.1

[12] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016. 3.2.1

[13] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016. 2.1, 2.2, 2.3.2, 2.4, 2.1, 2.2, 2.3

[14] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 2.2, 2.3.2

[15] Deniz A. Bezgin, Aaron B. Buhendwa, and Nikolaus A. Adams. Jax-fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows. *Computer Physics Communications*, page 108527, 2022. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2022.108527. URL https://www.sciencedirect.com/science/article/pii/S0010465522002466. 4.3.4

[16] Kiran S. Bhat, Steven M. Seitz, and Jovan Popovic. Computing the physical parameters of rigid-body motion from video. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *ECCV*, volume 2350 of *Lecture Notes in Computer Science*, pages 551–565. Springer, 2002. ISBN 3-540-43745-2. URL http://dblp.uni-trier.de/db/conf/eccv/eccv2002-1.html#BhatSP02. 3.2.1

[17] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023. 7.2

[18] Robert Bridson. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015. 4.1

[19] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 3.2.2

[20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 4.1, 4.2, 6.2

[21] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022. 1, 6.1, 6.3.3, 8.1, 8.2

[22] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. 6.1, 7.2

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165. 6.1, 6.2

[24] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 6.2

[25] A. Byravan and D. Fox. SE3-Nets: Learning rigid body motion using deep neural networks. *CoRR*, abs/1606.02378, 2016. 2.2

[26] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021. 6.2

[27] Stephanie C. Y. Chan, Adam Santoro, Andrew K. Lampinen, Jane X. Wang, Aaditya Singh, Pierre H. Richemond, Jay McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers, 2022. 6.2

[28] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. (document), 2.4, 2.6.3, 3.4.1, 3.3, 3.6.1, 3.6.1

[29] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *CoRR*, abs/1612.00341, 2016. URL http://arxiv.org/abs/1612.00341. 2.2

[30] Oscar Chang, Lampros Flokas, and Hod Lipson. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*, 2019. 3.2.2

[31] Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation. *arXiv preprint arXiv:2303.13873*, 2023. 6.2, 6.3.1, 6.3.1

[32] Tao Chen, Adithyavairavan Murali, and Abhinav Gupta. Hardware conditioned policies for multi-robot transfer learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9355–9366. Curran Associates Inc., 2018. 3.2.1

[33] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pages 297–307. PMLR, 2022. 4.2, 6.1, 7.1, 7.2

[34] Wei Chen, Heba Khamis, Ingvars Birznieks, Nathan F. Lepora, and Stephen J. Redmond. Tactile sensors for friction estimation and incipient slip detection—toward dexterous robotic manipulation: A review. *IEEE Sensors Journal*, 18(22):9049–9064, 2018. doi: 10.1109/JSEN.2018.2868340. 5.4

[35] Xuxin Cheng, Ashish Kumar, and Deepak Pathak. Legs as manipulator: Pushing quadrupedal agility beyond locomotion. *arXiv preprint arXiv:2303.11330*, 2023. 7.2

[36] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023. 1, 8.1, 8.2, 8.4, 8.6.1

[37] Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017. URL http:

//arxiv.org/abs/1704.02254. 2.2

[38] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. 3.4.2

[39] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks, 2019. 8.1

[40] Sammy Christen, Wei Yang, Claudia Pérez-D'Arpino, Otmar Hilliges, Dieter Fox, and Yu-Wei Chao. Learning human-to-robot handovers from point clouds, 2023. 6.3.3

[41] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136. 6.3.1

[42] Ignasi Clavera, Anusha Nagabandi, Ronald Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt: Meta-learning for model-based control. 03 2018. 3.1, 3.2.1, 3.3

[43] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014. 8.3, 8.8.4

[44] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016. 3.4.1, 4.2, 4.3.4, 6.2, 7.2

[45] Zichen Jeff Cui, Yibin Wang, Nur Muhammad, Lerrel Pinto, et al. From play to policy: Conditional behavior generation from uncurated robot data. *arXiv preprint arXiv:2210.10047*, 2022. 1

[46] Murtaza Dalal, Ajay Mandlekar, Caelan Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers, 2023. 6.2, 7.2

[47] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *CoRR*, abs/1910.11215, 2019. URL http://arxiv.org/abs/1910.11215. 2.2

[48] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31, 2018. 4.2

[49] Jean Decety and D. H. Ingvar. Brain structures participating in mental simulation of motor behavior: a neuropsychological interpretation. *Acta psychologica*, 73 1:13–34, 1990. 2.1

[50] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. *arXiv preprint arXiv:2212.08051*, 2022. 7.3.2

[51] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023. 7.3.2

[52] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *ArXiv*, abs/2105.05233, 2021. 8.2

[53] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017. 6.1

[54] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 7.4.2

[55] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023. 7.1, 7.2

[56] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. *arXiv preprint arXiv:2302.00111*, 2023. 7.2

[57] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and P. Abbeel. Learning universal policies via text-guided video generation. *ArXiv*, abs/2302.00111, 2023. 8.2

[58] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017. URL http://arxiv.org/abs/1710.05268. 2.1, 2.2

[59] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. 2.4, 2.4.1, 2.3, 3, 3.1, 3.2.1, 3.4.1

[60] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018. URL http://arxiv.org/abs/1812.00568. 2.2

[61] Yingruo Fan, Zhaojiang Lin, Jun Saito, Wenping Wang, and Taku Komura. Faceformer: Speech-driven 3d facial animation with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18770–18780, 2022. 5.5

[62] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation, 2019. 2.3.3

[63] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016. 2.2, 2.4.1, 3.1, 3.2.1, 3.4.1

[64] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016. 2.2

[65] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400. 3.1, 3.2.1, 3.3

[66] Adam Fishman, Adithyavairan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks, 2022. 6.2

[67] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *CoRR*, abs/1511.07404, 2015. 2.2, 3.1, 3.2.1, 3.4.1

[68] Zipeng Fu, Ashish Kumar, Ananye Agarwal, Haozhi Qi, Jitendra Malik, and Deepak Pathak. Coupling vision and proprioception for navigation of legged robots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17273–17283, 2022. 6.1

[69] Carl Gabbard. The role of mental simulation in embodied cognition. *Early Child Development and Care*, 183(5):643–650, 2013. 2.1

[70] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion, 2022. 6.4.1

[71] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

7.4.1, 7.6.3

[72] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020. 4.1, 4.2, 4.3.4, 6.2

[73] Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Michael Lingelbach, Aidan Curtis, Kevin Feigelis, Daniel M. Bear, Dan Gutfreund, David Cox, Antonio Torralba, James J. DiCarlo, Joshua B. Tenenbaum, Josh H. McDermott, and Daniel L. K. Yamins. Threedworld: A platform for interactive multi-modal physical simulation, 2021. 6.1, 7.2

[74] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Strip-stream: Integrating symbolic planners and blackbox samplers. *CoRR*, abs/1802.08705, 2018. URL http://arxiv.org/abs/1802.08705. 6.2

[75] Haoran Geng, Helin Xu, Chengyang Zhao, Chao Xu, Li Yi, Siyuan Huang, and He Wang. Gapartnet: Cross-category domain-generalizable object perception and manipulation via generalizable and actionable parts, 2023. 6.3.2, 6.4.4

[76] Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3d: Infinite resolution action detection transformer for robotic manipulation. *arXiv preprint arXiv:2306.17817*, 2023. (document), 3, 6.2, 8.1, 8.2, 3, 8.4

[77] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. Imagebind: One embedding space to bind them all. *arXiv preprint arXiv:2305.05665*, 2023. 7.2

[78] Nikolaos Gkanatsios, Ayush Jain, Zhou Xian, Yunchu Zhang, Christopher Atkeson, and Katerina Fragkiadaki. Energy-based models as zero-shot planners for compositional scene rearrangement. *arXiv preprint arXiv:2304.14391*, 2023. 6.2, 8.1, 8.2

[79] Ben Graham. Sparse 3d convolutional neural networks, 2015. 8.1

[80] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. *arXiv preprint arXiv:2302.04659*, 2023. 7.1, 7.2, 7.4.2

[81] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *Conference on Robot Learning*, pages 175–187. PMLR, 2023. 8.1, 8.1, 8.2, 8.3, 8.4.1, 8.5.1, 8.5.1, 8.5.2, 8.3, 1, 8.8.2, 8.8.3

[82] David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018. 2.2

[83] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 3.1, 3.2.2

[84] Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. *arXiv preprint arXiv:2307.14535*, 2023. 6.2, 7.2

[85] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 4.6.2, 7.3.3, 7.4.1, 7.6.3

[86] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H Huang, Dhruva Tirumala, Markus Wulfmeier, Jan Humplik, Saran Tunyasuvunakool, Noah Y Siegel, Roland Hafner, et al. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *arXiv preprint arXiv:2304.13653*, 2023. 7.1

[87] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019. 2.1, 2.2, 2.4, 2.3, 4, 3.1, 3.2.1, 3.4.1

[88] Jessica Hamrick, Peter Battaglia, and Joshua B Tenenbaum. Internal physics models guide probabilistic judgments about object dynamics. In *Proceedings of the 33rd annual conference of the cognitive science society*, pages 1545–1550. Cognitive Science Society Austin, TX, 2011. 3.1

[89] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001. 4.6.2

[90] Adam W Harley, Shrinidhi K Lakshmikanth, Fangyu Li, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Learning from unlabelled videos using contrastive predictive neural 3d mapping. *arXiv preprint arXiv:1906.03764*, 2019. 8.2

[91] Adam W Harley, Fangyu Li, Shrinidhi K Lakshmikanth, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Embodied view-contrastive 3d feature learning. *arXiv*, 2019. 2.1, 2.3.1

[92] Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. Multiple paired forward-inverse models for human motor learning and control. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 31–37. MIT Press, 1999. URL http://papers.nips.cc/paper/

1585-multiple-paired-forward-inverse-models-for-human-motor-learning-and-con
pdf. 3.1

[93] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870. 2.3.1, 3.3.1, 3.6.1

[94] Eric Heiden, Miles Macklin, Yashraj Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. Disect: A differentiable simulation engine for autonomous robotic cutting. *arXiv preprint arXiv:2105.12244*, 2021. 4.3.4, 7.2

[95] Eric Heiden, Christopher E. Denniston, David Millard, Fabio Ramos, and Gaurav S. Sukhatme. Probabilistic inference of simulation parameters via parallel differentiable simulation, 2022. 6.2

[96] Eric Heiden, Miles Macklin, Yashraj Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. Disect: A differentiable simulator for parameter inference and control in robotic cutting, 2022. 6.2

[97] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 8.2, 8.4, 8.4, 8.8.5

[98] Sebastian Höfer, Kostas E. Bekris, Ankur Handa, Juan Camilo Gamboa, Melissa Mozifian, Florian Golemo, Christopher G. Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Karen Liu, Jan Peters, Shuran Song, Peter Welinder, and Martha White. Sim2real in robotics and automation: Applications and challenges. *IEEE Trans Autom. Sci. Eng.*, 18(2):398–400, 2021. doi: 10.1109/TASE.2021.3064065. URL https://doi.org/10.1109/TASE.2021.3064065. 6.1

[99] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020. 4.3.4

[100] 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):1–14, 2018. 4.3.2

[101] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Fredo Durand. Difftaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2019. 4.2

[102] 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. 4.1, 4.2, 5.2

[103] Siyuan Huang, Zan Wang, Puhao Li, Baoxiong Jia, Tengyu Liu, Yixin Zhu, Wei Liang, and Song-Chun Zhu. Diffusion-based generation, optimization, and planning in 3d scenes. *ArXiv*, abs/2301.06015, 2023. 8.2

[104] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022. 6.2

[105] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608. 6.2

[106] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022. 7.2

[107] 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. *arXiv preprint arXiv:2307.05973*, 2023. 6.2, 7.1

[108] 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. 4.2, 4.3.2, 4.3.4

[109] Reina Ishikawa, Masashi Hamaya, Felix Von Drigalski, Kazutoshi Tanaka, and Atsushi Hashimoto. Learning by breaking: food fracture anticipation for robotic food manipulation. *IEEE Access*, 10:99321–99329, 2022. 5.4

[110] Hamid Izadinia, Qi Shan, and Steven M. Seitz. IM2CAD. *CoRR*, abs/1608.05137, 2016. 2.1

[111] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention, 2021. 6.3.3, 8.1, 8.2

[112] Stephen James and Andrew J Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *IEEE Robotics and Automation Letters*, 7(2):1612–1619, 2022. 8.3

[113] 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. (document), 7.1, 7.2, 7.3.1, 7.4.2, 8.1, 8.3, 8.5, 8.6.1

[114] Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748, 2022. 8.1, 8.2, 8.2

[115] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022. 8.2, 8.2

[116] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *CoRR*, abs/1812.10972, 2018. URL http://arxiv.org/abs/1812.10972. 2.1, 2.2, 2.3.2, 2.4, 2.6.3

[117] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022. 8.1, 8.2

[118] Yandong Ji, Gabriel B. Margolis, and Pulkit Agrawal. Dribblebot: Dynamic legged manipulation in the wild, 2023. 6.1

[119] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, and Dragomir Anguelov. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9644–9653, June 2023. 8.2

[120] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: Robot manipulation with multimodal prompts. 2023. 7.2

[121] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011. doi: 10.1109/ICRA.2011.5980391. 6.2

[122] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. *arXiv preprint arXiv:2303.05511*, 2023. 7.1

[123] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. Dall-e-bot: Introducing web-scale diffusion models to robotics. *IEEE Robotics and Automation Letters*, 8:3956–3963, 2022. 8.2

[124] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. Dall-e-bot: Introducing web-scale diffusion models to robotics. *IEEE Robotics and Automation Letters*, 2023. 7.2

[125] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal

motion planning. *The international journal of robotics research*, 30(7):846–894, 2011. 7.3.3

[126] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models. *arXiv preprint arXiv:2310.18308*, 2023. 5.5, 7.2

[127] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. *arXiv preprint arXiv:2006.05768*, 2020. 7.2

[128] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Mueller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620:982–987, 08 2023. doi: 10.1038/s41586-023-06419-4. 6.1

[129] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7.4.1, 7.6.3

[130] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018. 2.1, 2.2, 2.3.2

[131] Sylwester Klocek, Łukasz Maziarka, Maciej Wołczyk, Jacek Tabor, Jakub Nowak, and Marek Śmieja. Hypernetwork functional image representation. In *International Conference on Artificial Neural Networks*, pages 496–510. Springer, 2019. 3.2.2

[132] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017. 4.1, 4.2, 6.2

[133] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000. 8.3, 8.8.4

[134] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015. 2.1

[135] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021. 6.1

[136] Zihang Lai, Erika Lu, and Weidi Xie. Mast: A memory-augmented self-supervised tracker, 2020. 2.2

[137] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco

Hutter. Learning quadrupedal locomotion over challenging terrain. *CoRR*, abs/2010.11251, 2020. URL https://arxiv.org/abs/2010.11251. 6.1

[138] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection, 2016. 6.1

[139] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023. 7.1, 7.2

[140] Jue Kun Li, Wee Sun Lee, and David Hsu. Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems*, volume 14, pages 1–9, 2018. 3.1, 3.2.1, 3.4.1

[141] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023. 7.4.2

[142] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4):49, 2020. 7.5

[143] Sizhe Li, Zhiao Huang, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Contact points discovery for soft-body manipulations with differentiable physics. *arXiv preprint arXiv:2205.02835*, 2022. 4.1, 4.2

[144] Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. Gligen: Open-set grounded text-to-image generation. *arXiv preprint arXiv:2301.07093*, 2023. 6.3.1

[145] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018. 2.2, 3.2.1, 4.2

[146] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019. 2.1, 2.2

[147] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel L.K. Yamins, Jiajun Wu, Joshua B. Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models. In *International Conference on Machine Learning*, 2020. 2.1, 2.2

[148] 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, 2022. URL https://arxiv.org/abs/2209.07753. 6.2

[149] 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. *arXiv preprint arXiv:2209.07753*, 2022. 7.1, 7.2

[150] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 32, 2019. 4.2

[151] Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for text-to-behavior in minecraft, 2023. 6.2

[152] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 6.2

[153] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023. 7.2

[154] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 8.3

[155] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:2011.07215*, 2020. 4.1, 4.1, 4.2, 4.3.4, 6.2, 7.2

[156] Xingyu Lin, Zhiao Huang, Yunzhu Li, Joshua B Tenenbaum, David Held, and Chuang Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *arXiv preprint arXiv:2203.17275*, 2022. 4.1, 4.2, 4.8, 7.2, 7.3.3

[157] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, pages 256–266. PMLR, 2022. 4.2

[158] C Karen Liu and Dan Negrut. The role of physics-based simulators in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:35–58, 2021. 7.2

[159] Hao Liu, Lisa Lee, Kimin Lee, and Pieter Abbeel. Instruction-following agents with jointly pre-trained vision-language models. *arXiv preprint arXiv:2210.13431*, 2022. (document), 8.1, 8.1, 8.2, 8.3, 8.4.1, 8.4, 8.5.1, 8.5.1,

8.3, 2

[160] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D Plumbley. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023. 7.2

[161] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *CoRR*, abs/2107.13586, 2021. URL https://arxiv.org/abs/2107.13586. 6.2

[162] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object, 2023. 6.3.1, 6.3.1

[163] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. *arXiv preprint arXiv:2303.11328*, 2023. 7.3.2, 7.3.3

[164] Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-lambda: Disentangling dynamic task relationships. *arXiv preprint arXiv:2202.03091*, 2022. 8.1, 8.5.1, 8.3, 1, 8.6.1

[165] Weiyu Liu, Tucker Hermans, S. Chernova, and Chris Paxton. Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects. *ArXiv*, abs/2211.04604, 2022. 8.2

[166] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution, 2022. 8.2

[167] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019. 3.2.2

[168] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021. 7.2

[169] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 8.4.1

[170] Tiange Luo, Chris Rockwell, Honglak Lee, and Justin Johnson. Scalable 3d captioning with pretrained models. *arXiv preprint arXiv:2306.07279*, 2023. 7.6.1

[171] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic plan-

ning. *CoRR*, abs/1811.00090, 2018. URL http://arxiv.org/abs/1811.00090. 6.2

[172] Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):1–11, 2018. 4.2

[173] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023. 7.2, 7.7.3

[174] 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). 4.2

[175] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014. 4.1, 4.2, 4.3.4, 6.2, 7.2

[176] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. 6.4.2

[177] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021. 1, 8.1

[178] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015. URL http://arxiv.org/abs/1511.05440. 2.2, 3.2.1

[179] Michael James McDonald and Dylan Hadfield-Menell. Guided imitation of task and motion planning, 2021. 6.2, 7.2

[180] Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. Realfusion: 360 {\deg} reconstruction of any object from a single image. *arXiv preprint arXiv:2302.10663*, 2023. 7.2

[181] Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. Realfusion: 360 reconstruction of any object from a single image. In *CVPR*, 2023. URL https://arxiv.org/abs/2302.10663. (document), 6.2, 6.3.1, 6.3, 6.4.1

[182] Elliot Meyerson and Risto Miikkulainen. Modular universal reparameterization: Deep multi-task learning across diverse domains. In *Advances in Neural*

Information Processing Systems, pages 7903–7914, 2019. 3.2.2

[183] R. C. Miall and D. M. Wolpert. Forward models for physiological motor control. *Neural Netw.*, 9(8):1265–1279, November 1996. ISSN 0893-6080. doi: 10.1016/S0893-6080(96)00035-4. 2.2, 3.1

[184] Midjourney. Midjourney. https://www.midjourney.com/, 2022. 7.3.2

[185] Toki Migimatsu and Jeannette Bohg. Object-centric task and motion planning in dynamic environments. *CoRR*, abs/1911.04679, 2019. URL http://arxiv.org/abs/1911.04679. 6.2

[186] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021. 4.6.2

[187] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, 2018. 2.2, 3.2.1

[188] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. In *NIPS*, 2018. 2.1, 2.2

[189] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Adam Fishman, and Dieter Fox. Cabinet: Scaling neural collision detection for object rearrangement with procedural scene generation, 2023. 6.2, 7.2

[190] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl, 12 2018. 3.1, 3.2.1, 3.3

[191] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018. 3.3.3

[192] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019. 3.1, 3.2.1, 3.3, 3.4.1, 3.4.2, 3.5

[193] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation, 2022. 8.2

[194] Yashraj Narang, Balakumar Sundaralingam, Miles Macklin, Arsalan Mousavian, and Dieter Fox. Sim-to-real for robotic tactile sensing via physics-based simula-

tion and learned latent projections. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6444–6451. IEEE, 2021. 5.4

[195] Gautham Narasimhan, Kai Zhang, Ben Eisner, Xingyu Lin, and David Held. Self-supervised transparent liquid segmentation for robotic pouring. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4555–4561. IEEE, 2022. 4.2

[196] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images, 2019. 6.2

[197] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 8.2

[198] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*, 2015. 2.2, 3.2.1

[199] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015. 2.2

[200] OpenAI. Chatgpt. `https://openai.com/blog/chatgpt`, 2022. 7.3

[201] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 7.1, 7.2, 7.3.1

[202] OpenAI. Gpt-4 technical report, 2023. 6.3.1

[203] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019. 6.1

[204] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 7.4.2

[205] Priyam Parashar, Jay Vakil, Sam Powers, and Chris Paxton. Spatial-language attention policies for efficient robot learning. *arXiv preprint arXiv:2304.11235*, 2023. 8.1

[206] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control, 2022. 8.2

[207] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017. URL http://arxiv.org/abs/1705.05363. 2.2, 3.1, 3.2.1

[208] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *ICLR*, 2018. 2.4.1

[209] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, and Sam Devlin. Imitating human behaviour with diffusion models, 2023. 8.2

[210] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020. 4.2

[211] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2161–2168. IEEE, 2017. 3.4.1

[212] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. *CoRR*, abs/1604.01360, 2016. URL http://arxiv.org/abs/1604.01360. 2.2

[213] Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 425–435, 2018. 3.2.2

[214] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 6.2, 6.3.1, 6.3.1, 6.4.1, 7.2

[215] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*, 2020. 4.3.4

[216] 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. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020. 6.1

[217] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In

International conference on machine learning, pages 8748–8763. PMLR, 2021. 7.4.2, 8.2, 8.3, 8.4.1, 8.5.2

[218] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022. 6.1

[219] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Learning humanoid locomotion with transformers. *arXiv preprint arXiv:2303.03381*, 2023. 7.2

[220] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022. 8.2

[221] Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. In *International Conference on Machine Learning*, pages 5361–5369, 2019. 3.2.2

[222] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. 8.2

[223] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019. 7.6.1

[224] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023. 8.1, 8.2

[225] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes, 2023. 6.3.1

[226] Lukasz Romaszko, Christopher K. I. Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *ICCV Workshops*, Oct 2017. 2.1

[227] Lukasz Romaszko, Christopher KI Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 851–859, 2017. 2.1

[228] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2022. 8.2

[229] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and

Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 6.1, 6.3.1, 7.1

[230] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 6.3.1

[231] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, abs/2205.11487, 2022. 8.2

[232] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv:1806.01242*, 2018. 2.1, 2.2, 3.2.1

[233] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479, 2018. 3.1, 3.2.1, 3.4.2

[234] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks, 02 2020. 2.1, 2.2

[235] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020. 4.2

[236] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research, 2019. 6.1

[237] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335. PMLR, 2018. 4.1, 4.1, 4.2

[238] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 4.6.2, 6.3.3, 6.4.2, 7.3.3

[239] Daniel Seita, Pete Florence, Jonathan Tompson, Erwin Coumans, Vikas Sindhwani, Ken Goldberg, and Andy Zeng. Learning to rearrange deformable cables,

fabrics, and bags with goal-conditioned transporter networks, 2021. 8.1, 8.2

[240] Daniel Seita, Yufei Wang, Sarthak Shetty, Yao Li, Zackory Erickson, and David Held. Toolflownet: Robotic manipulation with tools via predicting tool flow from point clouds. In *Conference on Robot Learning*. PMLR, 2022. 4.1, 4.2

[241] Daniel Seita, Yufei Wang, Sarthak J Shetty, Edward Yao Li, Zackory Erickson, and David Held. Toolflownet: Robotic manipulation with tools via predicting tool flow from point clouds. In *Conference on Robot Learning*, pages 1038–1049. PMLR, 2023. 7.2

[242] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE, 2018. 4.1

[243] Joan Serrà, Santiago Pascual, and Carlos Segura Perales. Blow: a single-scale hyperconditioned flow for non-parallel raw-audio voice conversion. *Advances in Neural Information Processing Systems*, 32:6793–6803, 2019. 3.2.2

[244] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022. 8.2

[245] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018. 8.2

[246] Bokui Shen, Xinchen Yan, Charles R. Qi, Mahyar Najibi, Boyang Deng, Leonidas Guibas, Yin Zhou, and Dragomir Anguelov. Gina-3d: Learning to generate implicit neural assets in the wild, 2023. 6.2

[247] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. *ArXiv*, abs/2109.12098, 2021. 8.1

[248] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022. 7.4.3, 8.2, 8.2

[249] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023. (document), 8.1, 8.1, 8.2, 8.2, 8.3, 8.4.1, 8.5.1, 8.5, 8.5.1, 8.8.2, 8.7, 8.8.6, 8.8.7

[250] Zilin Si and Wenzhen Yuan. Taxim: An example-based simulation model for gelsight tactile sensors. *IEEE Robotics and Automation Letters*, 7(2):2361–2368, 2022. 5.4

[251] Zilin Si, Zirui Zhu, Arpit Agarwal, Stuart Anderson, and Wenzhen Yuan. Grasp stability prediction with sim-to-real transfer from tactile sensing. In *2022*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 7809–7816, 2022. doi: 10.1109/IROS47612.2022.9981863. 5.4

[252] Zilin Si, Gu Zhang, Qingwei Ben, Branden Romero, Zhou Xian, Chao Liu, and Chuang Gan. Difftactile: A physics-based differentiable tactile simulator for contact-rich robotic manipulation. *arXiv preprint arXiv:2403.08716*, 2024. 5.4

[253] Maximilian Sieb, Zhou Xian, Audrey Huang, Oliver Kroemer, and Katerina Fragkiadaki. Graph-structured visual imitation. In *Conference on Robot Learning*, pages 979–989. PMLR, 2020. 1, 6.1, 8.1

[254] Jascha Narain Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *ArXiv*, abs/1503.03585, 2015. 8.2

[255] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 8.2

[256] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023. 7.2

[257] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments, 2021. 6.1

[258] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Conference on Robot Learning*, pages 477–490. PMLR, 2022. 7.1, 7.2, 7.4.2

[259] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999. 4.3.2

[260] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013. 4.3.2

[261] 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. 8.2

[262] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv*

preprint arXiv:2104.09864, 2021. 8.1, 8.3, 8.3, 8.8.4

[263] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2022. 8.2

[264] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham. Can robots assemble an ikea chair? *Science Robotics*, 3(17):eaat6385, 2018. 4.2

[265] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. 7.6.3, 8.3, 8.8.4

[266] Sudharshan Suresh, Zilin Si, Joshua G Mangelson, Wenzhen Yuan, and Michael Kaess. Shapemap 3-d: Efficient shape mapping through dense touch and vision. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7073–7080. IEEE, 2022. 5.4

[267] Junshu Tang, Tengfei Wang, Bo Zhang, Ting Zhang, Ran Yi, Lizhuang Ma, and Dong Chen. Make-it-3d: High-fidelity 3d creation from a single image with diffusion prior, 2023. (document), 6.2, 6.3, 6.4.1

[268] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023. 7.3

[269] Yuval Tassa, Tom Erez, and William D Smart. Receding horizon differential dynamic programming. In *Advances in neural information processing systems*, pages 1465–1472, 2008. 2.3.3

[270] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. 7.3.2, 7.6.1

[271] 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 *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017. 6.3.1, 7.5

[272] Joshua 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. *CoRR*, abs/1703.06907, 2017. URL `http://arxiv.org/abs/1703.06907`. 2.6.3

[273] 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. 3.4.2, 4.2, 4.3.4, 6.2, 7.2

[274] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1930–1936. AAAI Press, 2015. ISBN 9781577357384. 6.2

[275] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 7.1, 7.2

[276] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. *arXiv:1901.00003*, 2018. 2.1, 2.3.1, 2.6.4, 3.3.1, 3.4.1, 3.6.1

[277] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2595–2603, 2019. 8.2

[278] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators. *arXiv preprint arXiv:2011.06464*, 2020. 1, 2, 3.5, 4.2, 6.2, 8.1

[279] Julen Urain, Niklas Funk, Georgia Chalvatzaki, and Jan Peters. Se (3)-diffusionfields: Learning cost functions for joint grasp and motion optimization through diffusion. *arXiv preprint arXiv:2209.03855*, 2022. 8.1, 8.4

[280] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 8.4

[281] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019. 3.2.2

[282] Chen Wang, Linxi Fan, Jiankai Sun, Ruohan Zhang, Li Fei-Fei, Danfei Xu, Yuke Zhu, and Anima Anandkumar. Mimicplay: Long-horizon imitation learning by watching human play. *arXiv preprint arXiv:2302.12422*, 2023. 1

[283] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. 6.2

[284] Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A. Yeh, and Greg Shakhnarovich. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation, 2022. 6.3.1

[285] Kun Wang, William R. Johnson III au2, Shiyang Lu, Xiaonan Huang, Joran Booth, Rebecca Kramer-Bottiglio, Mridul Aanjaneya, and Kostas Bekris. Real2sim2real transfer for control of cable-driven robots via a differentiable physics engine, 2023. 6.2

[286] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. In *Arxiv*, 2023. (document), 7.2, 7.1, 7.4.2, 7.4.3

[287] Tsun-Hsuan Wang, Pingchuan Ma, Andrew Everett Spielberg, Zhou Xian, Hao Zhang, Joshua B Tenenbaum, Daniela Rus, and Chuang Gan. Softzoo: A soft robot co-design benchmark for locomotion in diverse environments. *arXiv preprint arXiv:2303.09555*, 2023. 5.1, 5.3, 6.2

[288] Yen-Jen Wang, Bike Zhang, Jianyu Chen, and Koushil Sreenath. Prompt a robot to walk with large language models. *arXiv preprint arXiv:2309.09969*, 2023. 7.1, 7.2

[289] Yian Wang, Xiaowen Qiu, Jiageng Liu, Zhehuan Chen, Jiting Cai, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. Architect: Generating vivid and interactive 3d scenes with hierarchical 2d inpainting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 5.5

[290] Yufei Wang, Zhanyi Sun, Zackory Erickson, and David Held. One policy to dress them all: Learning to dress people with diverse poses and garments. *arXiv preprint arXiv:2306.12372*, 2023. 7.2

[291] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023. 2, 5.5

[292] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015. 3.1, 3.2.1

[293] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022. URL https://arxiv.org/abs/2201.11903. 6.2

[294] Thomas Weng, Sujay Man Bajracharya, Yufei Wang, Khush Agrawal, and David Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, pages 192–202. PMLR, 2022. 7.1, 7.2

[295] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum.

Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 127–135. Curran Associates, Inc., 2015. 2.2

[296] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017. 2.2, 2.4, 1

[297] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017. 2.2, 3.4.1

[298] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *arXiv preprint arXiv:2305.05658*, 2023. 7.2

[299] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling, 2022. 8.2

[300] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations. *arXiv preprint arXiv:1910.13439*, 2019. 4.2

[301] Zhou Xian, Shamit Lal, Hsiao-Yu Tung, Emmanouil Antonios Platanios, and Katerina Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. *arXiv preprint arXiv:2103.09439*, 2021. 1, 6.2, 8.2

[302] Zhou Xian, Theophile Gervet, Zhenjia Xu, Yi-Ling Qiao, and Tsun-Hsuan Wang. Towards a foundation model for generalist robots: Diverse skill learning at scale via automated task and scene generation. *arXiv preprint arXiv:2305.10455*, 2023. 6.1, 7.1

[303] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. *Conference on Robot Learning*, 2023. 6.2

[304] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *7th Annual Conference on Robot Learning*, 2023. 3

[305] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. *arXiv preprint arXiv:2303.02346*,

2023. 1, 5.1, 5.3, 6.2, 7.2, 7.3.3

[306] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020. 4.1, 4.2, 4.3.4, 5.5, 6.1, 6.2, 6.3.2, 6.4.3, 6.4.4, 7.2, 7.3.1, 7.3.2

[307] Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Regression planning networks. *CoRR*, abs/1909.13072, 2019. URL http://arxiv.org/abs/1909.13072. 6.2

[308] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. *arXiv preprint arXiv:2107.07501*, 2021. 4.1, 4.2

[309] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*, 2022. 4.1, 4.2, 4.8

[310] Jie Xu, Sangwoon Kim, Tao Chen, Alberto Rodriguez Garcia, Pulkit Agrawal, Wojciech Matusik, and Shinjiro Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *Conference on Robot Learning*, pages 1488–1498. PMLR, 2023. 5.4

[311] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B. Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *CoRR*, abs/1906.03853, 2019. URL http://arxiv.org/abs/1906.03853. 3.1, 3.2.1, 3.4.1

[312] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019. 3.1, 3.2.1, 3.3.1

[313] Zhenjia Xu, Zhou Xian, Xingyu Lin, Cheng Chi, Zhiao Huang, Chuang Gan, and Shuran Song. Roboninja: Learning an adaptive cutting policy for multi-material objects. *arXiv preprint arXiv:2302.11553*, 2023. 5.1, 5.3, 7.1, 7.2, 7.3.3, 7.5

[314] Ben Yang, Youquan Liu, Lihua You, and Xiaogang Jin. A unified smoke control method based on signed distance field. *Computers & graphics*, 37(7):775–786, 2013. 4.2

[315] Yu-Qi Yang, Yu-Xiao Guo, Jian-Yu Xiong, Yang Liu, Hao Pan, Peng-Shuai Wang, Xin Tong, and Baining Guo. Swin3d: A pretrained transformer backbone for 3d indoor scene understanding, 2023. 8.2

[316] Yufei Ye, Dhiraj Gandhi, Abhinav Gupta, and Shubham Tulsiani. Object-centric

forward modeling for model predictive control, 2019. 2.4.1

[317] Yufei Ye, Dhiraj Gandhi, Abhinav Gupta, and Shubham Tulsiani. Object-centric forward modeling for model predictive control. *the Conference on Robot Learning (CoRL)*, 2019. 2.1, 2.2, 2.3.2, 2.4, 2.1, 2.2, 2.3

[318] Lin Yen-Chen, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. Learning to see before learning to act: Visual pre-training for manipulation, 2021. 8.2

[319] Sarah Young, Dhiraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. Visual imitation made easy. In *Conference on Robot Learning*, pages 1992–2005. PMLR, 2021. 1

[320] Kuan-Ting Yu, Maria Bauzá, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. *CoRR*, abs/1604.04038, 2016. (document), 2.4, 2.6.3, 3.4.1, 3.3, 3.6.1

[321] Kuan–Ting Yu and Alberto Rodriguez. Realtime state estimation with tactile and visual sensing. application to planar manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7778–7785, 2018. doi: 10.1109/ICRA.2018.8463183. 5.4

[322] 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. 7.2, 7.4.2

[323] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Jodilyn Peralta, Brian Ichter, et al. Scaling robot learning with semantically imagined experience. *arXiv preprint arXiv:2302.11550*, 2023. 7.2

[324] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023. 6.2, 7.2

[325] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *CoRR*, abs/1903.11239, 2019. 2.2

[326] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020. 4.2

[327] Andy Zeng, Peter R. 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. In *Conference on Robot Learning*, 2020. 8.1

[328] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020. 7.2

[329] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021. 8.1, 8.2

[330] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018. 3.2.2

[331] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020. 8.4.1

[332] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv preprint arXiv:2302.05543*, 2023. 6.3.1

[333] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pages 7444–7453, 2019. 3.1, 3.2.1, 3.4.1

[334] Xiaoshuai Zhang, Rui Chen, Ang Li, Fanbo Xiang, Yuzhe Qin, Jiayuan Gu, Zhan Ling, Minghua Liu, Peiyu Zeng, Songfang Han, et al. Close the optical sensing domain gap by physics-grounded active stereo sensor simulation. *IEEE Transactions on Robotics*, 2023. 7.5

[335] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. *ArXiv*, abs/2210.17366, 2022. 8.2

[336] Wenyang Zhou, Zhiyang Dou, Zeyu Cao, Zhouyingcheng Liao, Jingbo Wang, Wenjia Wang, Yuan Liu, Taku Komura, Wenping Wang, and Lingjie Liu. Emdm: Efficient motion diffusion model for fast and high-quality motion generation. In *European Conference on Computer Vision*, pages 18–38. Springer, 2025. 5.5

[337] Bo Zhu, Michiaki Iwata, Ryo Haraguchi, Takashi Ashihara, Nobuyuki Umetani, Takeo Igarashi, and Kazuo Nakazawa. Sketch-based dynamic illustration of fluid systems. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–8, 2011. 4.2

[338] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory, 2023. 6.2

[339] Xupeng Zhu, Dian Wang, Ondrej Biza, Guanang Su, Robin Walters, and Robert Platt. Sample efficient grasp learning using equivariant models, 2022. 8.1

[340] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 1097–1100, 2018. 7.4.2

[341] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. *CoRR*, abs/2012.07277, 2020. URL https://arxiv.org/abs/2012.07277. 6.2

[342] Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023. 7.1, 7.2