TRAINING AND DEPLOYING VISUAL AGENTS AT SCALE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Linxi Fan

August 2021

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Fei-Fei Li, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Juan Carlos Niebles Duque

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Jiajun Wu

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Autonomous agents that perceive and interact with the world, such as home robots and self-driving vehicles, hold great promises to a future that automates mundane tasks and improves the living standards for billions of people. However, two major obstacles stand in our way towards this grand goal. First, modern AI systems require huge amount of data to learn meaningful behaviors, yet training them directly on physics robots is unscalable due to high cost and low efficiency. Second, mobile robot platforms typically have limited onboard computing resources but demand low reaction latency, which hinders the mass deployment of large-capacity visual models.

In this dissertation, we will explore an effective recipe towards developing algorithms and systems that are able to train and deploy visual agents at scale. The key idea is to train the agents in rich simulation, then overcome the sim-to-real gap, and finally deploy efficiently on edge devices with lightweight video processing architectures.

This dissertation is organized around 4 primary components in the pipeline. First, we propose an open-source distributed framework that provides a full-stack solution to accelerate reinforcement learning (RL) significantly for complex robotics tasks. Second, we construct an ecologically valid and visually realistic simulator for home robotic tasks. Third, we introduce a novel policy learning method that achieves zero-shot generalization to unseen visual environments with large distributional shifts, which facilitates sim-to-real transfer. Finally, we design a new family of video learning architectures that enables deep video understanding for visual agents on resource-constrained devices.

We hope that the techniques and ideas presented in this dissertation will bring us one step closer to the future where intelligent robots will become as ubiquitous as smartphones in our lives.

Acknowledgments

First and foremost, I am deeply indebted to my doctoral advisor, Fei-Fei Li, who has significantly reshaped my career and perspective as an aspiring scientist over the past five years. There are so many things I learned from Fei-Fei during the countless hours I spent under her mentorship - two of which I would like to highlight in particular. First, Fei-Fei has taught me that research without a north star is like a body without soul. Her methodology guides me towards not just solving the problems, but finding problems that are *worth solving*. She once told me her favorite quote by Albert Einstein: "The mere formulation of a problem is often far more essential than its solution, which [...] requires creative imagination and marks real advances in science." Over the years, I acquired better and better research taste under her guidance, which has been a truly transformative journey personally. Second, Fei-Fei has made me realize that mentoring students is an extremely rewarding experience and a rich learning opportunity in itself, particularly with mentees from under-represented demographics. By her encouragement, I mentored more than a dozen undergraduate and master students at Stanford. I have come to understand that empowering others is a beautiful way to improve myself as a stronger scientist and a better human being. Fei-Fei will continue to be my role model as a prolific researcher, a thought leader, and an exceptional educator. I could not possibly dream of a better Ph.D. advisor.

I would like to thank my Ph.D. Oral Defense committee, Serena Yeung (Oral Chair), Jiajun Wu, Juan Carlos Niebles, and Silvio Savarese. It is my greatest honor and privilege to be mentored by many other alumni and faculty members at Stanford as well: Yuke Zhu, Animesh Garg, Roberto Martín-Martín, Emma Brunskill, Percy Liang, Justin Johnson, Andrej Karpathy, Tianlin "Tim" Shi, and Andre Esteva. Thank you for helping me define and redefine my research direction, and giving me a wealth of inspiration and motivation to carry on.

At Stanford, I have been very fortunate to collaborate with a group of brilliant minds, including Shyamal Buch, Guanzhi Wang, Agrim Gupta, Kaylee Burns, Fei Xia, Bokui "Will" Shen, De-An Huang, Chengshu Li, Claudia Pérez-D'Arpino, Sanjana Srivastava, Josiah Wong, Ademi Adeniji, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, and Joan Creus-Costa. I could not have completed so much research work without them. Thank you all so much for making the deadlines less daunting, and my research life more enjoyable and fruitful.

In addition, I have met many wonderful colleagues and made lots of close friends at Stanford Vision Lab, other labs at Stanford, as well as outside the university – Shikun Liu, Zhiyuan "Jerry" Lin, Ajay Mandlekar, Kuan Fang, Damian Mrowca, Zelun "Alan" Luo, Jingwei Ji, Danfei Xu, Chien-Yi Chang, Michelle Lee, Ranjay Krishna, Andrey Kurenkov, Boxiao Pan, Bingbin Liu, Bohan Wu, Chen Wang, Shuran Song, Michelle Guo, Christopher Choy, Lyne P. Tchapmi, Kaichun Mo, Jiaming Song, Shengjia Zhao, Ji Lin, Song Han, Cewu Lu, Yanan Sui, Yang Song, Jian Zhang, Peng Qi, Yu Yan, Jiwei Li, Kexin Rong, Helen Roman, JunYoung Gwak, Kevin Chen, Suraj Nair, Albert Haque, Yanjun Chen, Ryan Cao, Amir Zamir, Iro Armeni, Timnit Gebru, and Judy Hoffman. This is by no means an exhaustive list of individuals who gave me warm support and fond memory during graduate school. I am deeply grateful for their valuable presence in this chapter of my life.

Furthermore, I had the great fortune of working with fantastic researchers at industry labs during summer internships. During summer of 2020, I worked with Anima Anandkumar, Zhiding Yu, Jean Kossaifi, Jacob Austin, Huaizu Jiang, Weili Nie, and Chaowei Xiao at NVIDIA Research. In the summer of 2018, I collaborated with Lu Jiang, Jia Li, and Mei Han at Google Cloud AI. I deeply cherish my time at the company labs exploring different research venues and broadening my horizon.

I would also like to extend special thanks to the advisors during my undergraduate years: Michael Collins, Yoshua Bengio, Andrew Ng, Dario Amodei, Adam Coates, Shree Nayar, and Venkat Venkatasubramanian. Their inspiring works and mentorship were the primary reason that convinced me to pursue a Ph.D. degree. I would not be writing this page if they were not there at the critical junctions of my life.

Most importantly, I could not have completed this journey without the unconditional love and unwavering support from my parents, Shelly and James. Even though they had no formal training in scientific fields, they took a genuine interest in my work. It has always been an absolute joy to explain my progress and difficulties to them. Thank you both for being the most awesome listeners! Finally, I would like to express my deepest gratitude towards my girlfriend, Ying, who is and will always be the greatest gift that my time at Stanford has given me. Thank you for believing in me at times when I don't even believe in myself.

Learn the rules like a pro,
so you can break them like an artist.

— Pablo Picasso

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Autonomous agents that perceive and interact with the world hold great promises to a future that automates mundane tasks and improves the living standards for billions of people. Over the years, researchers have made great progress in visual policy learning on robot platforms, from self-driving cars [255, 217] to autopilot drones [10, 2, 150], from robot workers in warehouse [151] to robot chefs at home [144] (Fig. 1.1). However, the dream of intelligent robots roaming everywhere is still out of our reach. In this dissertation, we identify two major obstacles that stand in our way towards reaching this grand goal.

First, modern AI systems require huge amount of data to learn meaningful behaviors [107, 42, 139, 142, 226, 197, 227]. Supervised computer vision problems, such as image classification [42, 182, 77] and object detection [178, 76], have witnessed tremendous breakthroughs thanks to the large-scale and high-quality datasets like ImageNet [42] and MS-COCO [128].

Vision-based robotics, however, do not enjoy the same luxury of real and human-annotated datasets, because embodied agents are required to explore the environment in order to learn from a tight vision-and-action feedback loop [194, 68, 69, 192, 140, 142]. Unfortunately, training directly on physical robots can be very difficult. To name a few major roadblocks:

- **Low efficiency** [119, 248, 25, 66, 165]: the experiment iteration can be very slow, as throttled by the physical control frequency and agility of the commercially available robots in lab settings. This presents a serious bottleneck to rapidly prototyping new algorithms and tuning hyperparameters.
- **Costly setup** [121, 154]: robot arms can be very expensive, sometimes exceeding hundreds of thousands of dollars. The hefty price tag dramatically raises the barrier to entry for small-budget researchers. Moreover, it could be potentially dangerous to let the robot freely explore

the physical environment. The mechanical parts could break and incur extra monetary cost to replace. Even worse, a nearby human observer could be injured as a result of any faulty motions.

- **Manual efforts**: it takes nontrivial human efforts to set and reset the physical environment for the agent to interact with. Limited by the lab setting and scarce human labor, the agent will not be able to experience nearly as much diversity as the uncontrolled physical world out there.

These factors combined preclude us from diverse exploration and large-scale experimentation directly on physical platforms.

Second, in addition to the difficulties in training, deployment presents an equally daunting set of challenges. The visual world is not a still image. In fact, it is a continuous, ever-changing video stream, which is notoriously expensive to process [102, 199, 232]. To understand the events happening around it, an agent must be able to perform video inference very efficiently [222, 127, 246, 169]. This is exacerbated by the limited computing resources on mobile robot platforms, combined with the short latency that is required of the agent to react in real time to its surroundings [46]. These compounding difficulties greatly hinder the mass deployment of large-capacity visual models to real robot applications.

In this dissertation, we will explore an effective recipe towards developing algorithms and systems that are able to train and deploy visual agents at scale. The key idea is to train the agents in rich simulation, then overcome the sim-to-real gap, and finally deploy efficiently on edge devices with lightweight video processing architectures. This dissertation is organized around 4 primary components in the pipeline.

First, we propose an open-source distributed framework [208, 52] that provides a full-stack solution to accelerate reinforcement learning (RL) significantly for complex robotics tasks. Second, we construct an ecologically valid and visually realistic simulator for home robotic tasks [195, 241, 243]. Training in simulation enables us to sidestep all the difficulties of learning directly on physical robots:

- **High efficiency and scalability** [148, 9, 34, 85, 49, 124, 123]: armed with the distributed RL framework, we are able to massively scale up training on modern cloud platforms. The throughput we can achieve in simulation is equivalent to days of real world interaction in a matter of seconds. This high volume of experience satisfies the data hunger of the state-of-the-art deep RL algorithms. We demonstrate significant advancements in the performance of our simulated robots.

- **Safe exploration**: thanks to the fully simulated environments, we no longer have the issue of dangerous exploration in the physical world. The robot agents can freely carry out experimentation of novel behaviors, supported by the high-fidelity physics engine in the simulator.

Figure 1.1: Autonomous visual agents are playing increasingly important roles in our lives. In clockwise order: Tesla Autopilot [217], Jetson autonomous drone [150], Moley robotic kitchen demonstration [144], and IsaacSim robotic warehouse workers [151].

- **Diverse experience** [195, 105, 188, 189, 60]: we control every aspect of the simulation, which means that we can add novel scenes, objects, appearances, and interaction modes at scale. These variations greatly contribute to the diversity of experience that the embodied agent receives, which gives rise to enormous improvements in the learning performance. Such variety would not have been easily achieveable in the physical world due to budget and space constraints.

Third, we introduce a novel policy learning method [51] that achieves zero-shot generalization to unseen visual environments with large distributional shifts. This step enables the agent trained purely in simulation to adapt to the real world rapidly and robustly [72, 218]. Finally, we design a new family of video learning architectures [50] that pushes forward the Pareto frontier of efficiency and accuracy, which unlocks deep video understanding [127, 222, 169, 246] for visual agents on resource-constrained devices.

We hope that the techniques and ideas presented in this dissertation will bring us one step closer to the future where intelligent robots will become as ubiquitous as smartphones in our lives.

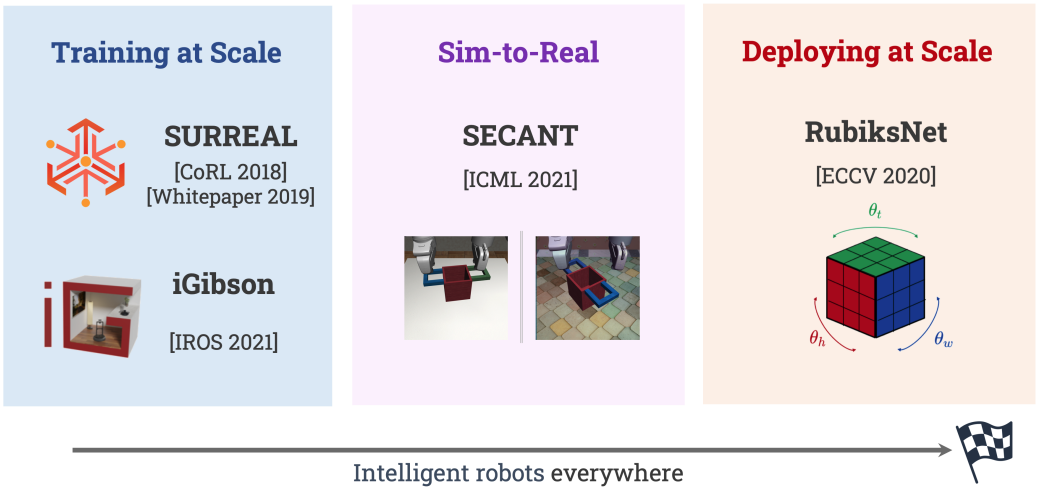


Figure 1.2: Overview of our proposed recipe for training and deploying visual agents at scale. Each chapter in this dissertation represents a key component in the pipeline, which we hope will bring us closer to the goal of deploying intelligent robots in every aspect of our lives.

1.2 Thesis Outline

Based on the proposed recipe of training and deploying visual agents at scale, we present an outline of this dissertation (Figure 1.2):

Chapter 2 introduces SURREAL [208], an open-source scalable framework that supports state-of-the-art distributed reinforcement learning algorithms. Reproducibility has been a significant challenge in deep reinforcement learning and robotics research. Open-source frameworks and standardized benchmarks can serve an integral role in rigorous evaluation and reproducible research. SURREAL represents a major effort in this direction. We design a principled distributed learning formulation that accommodates both on-policy and off-policy learning. We demonstrate that SURREAL algorithms outperform existing open-source implementations in both agent performance and learning efficiency.

Chapter 3 takes a deeper dive into the system design aspect of SURREAL [52], which powers the framework's high performance and massive scalability. SURREAL-System consists of a stack of four layers: Provisioner, Orchestrator, Protocol, and Algorithms. The Provisioner abstracts away the machine hardware and node pools across different cloud providers. The Orchestrator provides a unified interface for scheduling and deploying distributed algorithms by high-level description, which is capable of deploying to a wide range of hardware from a personal laptop to full-fledged cloud clusters. The Protocol provides network communication primitives optimized for RL. Finally, the SURREAL algorithms, such as Proximal Policy Optimization (PPO) [194] and Evolution Strategies (ES) [186], can easily scale to 1000s of CPU cores and 100s of GPUs. The learning performances of

our distributed algorithms establish new state-of-the-art on OpenAI Gym [15] and Robotics Suites tasks [208].

Chapter 4 presents iGibson 1.0 [195], a novel simulation environment to develop robotic solutions for interactive tasks in large-scale realistic scenes. Our environment contains 15 *fully interactive* home-sized scenes with 108 rooms populated with rigid and articulated objects. The scenes are replicas of real-world homes, with distribution and the layout of objects aligned to those of the real world. iGibson 1.0 integrates several key features to facilitate the study of interactive tasks: (1) generation of high-quality virtual sensor signals (RGB, depth, segmentation, LiDAR, flow and so on), (2) domain randomization to change the materials of the objects (both visual and physical) and/or their shapes, (3) integrated sampling-based motion planners to generate collision-free trajectories for robot bases and arms, and (4) intuitive human-iGibson interface that enables efficient collection of human demonstrations. Through experiments, we show that the full interactivity of the scenes enables agents to learn useful visual representations that accelerate the training of downstream manipulation tasks. We also show that iGibson features enable the generalization of navigation agents, and that the human-iGibson interface and integrated motion planners facilitate efficient imitation learning of human demonstrated (mobile) manipulation behaviors.

Chapter 5 proposes SECANT [51], a novel algorithm that facilitates sim-to-real transfer. Generalization has been a long-standing challenge for reinforcement learning (RL). Visual RL, in particular, can be easily distracted by irrelevant factors in high-dimensional observation space. In this chapter, we consider robust policy learning which targets zero-shot generalization to unseen visual environments with large distributional shift. SECANT is a novel self-expert cloning technique that leverages image augmentation in two stages to *decouple* robust representation learning from policy optimization. Specifically, an expert policy is first trained by RL from scratch with *weak* augmentations. A student network then learns to mimic the expert policy by supervised learning with *strong* augmentations, making its representation more robust against visual variations compared to the expert. Extensive experiments demonstrate that SECANT significantly advances the state of the art in zero-shot generalization across 4 challenging domains: Deepmind Control [214], autonomous driving under diverse weathers [46], robotic manipulation in distracting scenarios [266], and indoor object navigation in rooms with novel layouts and interior decorations [195].

Chapter 6 discusses RubiksNet [50], an efficient video inference architecture that unlocks deep temporal understanding on mobile robot platforms. Video recognition is a complex task dependent on modeling spatial and temporal context. Standard approaches rely on 2D or 3D convolutions to process such context, resulting in expensive operations with millions of parameters. Recent efficient architectures leverage a channel-wise shift-based primitive as a replacement for temporal convolutions, but remain bottlenecked by spatial convolution operations to maintain strong accuracy and a fixed-shift scheme [127, 238, 261]. Naively extending such developments to a 3D setting is a difficult, intractable goal. To this end, we propose RubiksNet, based on a novel *learnable*

3D spatiotemporal shift operation instead. We analyze the suitability of our new primitive for video action recognition and explore several novel variations of our approach to enable stronger representational flexibility while maintaining an efficient design. We benchmark our approach on several standard video recognition datasets, and observe that our method achieves comparable or better accuracy than prior work on efficient video action recognition at a fraction of the runtime latency and memory footprint. We also perform a series of controlled ablation studies to verify our significant boost in the efficiency-accuracy tradeoff curve is rooted in the core contributions of our RubiksNet architecture.

Finally, chapter 7 provides a summary of the key ideas explored in this dissertation, and discusses promising future ideas for scaling up visual robot learning even further.

Chapter 2

Scalable Distributed Training for Visual Agents

2.1 Introduction

Reinforcement learning (RL) has been an established framework in robotics to learn controllers via trial and error [104]. Classic reinforcement learning literature in robotics has largely relied on hand-crafted features and shallow models [164, 163]. The recent success of deep neural networks in learning representations [112] has incentivized researchers to use them as powerful function approximators to tackle more complex control problems, giving rise to *deep reinforcement learning* [142, 126]. Prior work has explored the potentials of using deep RL for robot manipulation [119, 248]. Recently we have witnessed an increasing number of successful demonstrations of deep RL in simulated environments [183, 265, 161] and on real hardware [25, 66].

However, the challenges of reproducibility and replicability in deep RL have impaired research progress [79]. Today's deep RL research has not been as accessible as it should be. Reproducing and validating published results is rarely straightforward, as it can be affected by numerous factors such as hyperparameter choices, initializations, and environment stochasticity, especially in absence of open-source code releases. Furthermore, owing to the data-hungry nature of deep RL algorithms, we have observed a rising number of state-of-the-art results being achieved by highly sophisticated distributed learning systems [197, 85, 49]. The heavy engineering factor in cutting-edge deep RL research has increased the barrier of entry even more for new researchers and small labs.

Meanwhile, benchmarking and reproducibility of robotics research have also been a long-standing challenge in the robotics community [41, 131]. Attempts to improve this situation include annual robotic competitions [7, 37] and standardized object sets [19]. However, designing robotic benchmarks that can be both standardized and widely accessible remains to be an open problem.

To further complicate the situation, many deep RL papers do not open-source their codebases. Devil is in the details of RL implementation, because seemingly small changes can lead to dramatic degradation of performance. The absence of official code releases leads to a dichotomy between impressive results reported in published papers and the frustration of using DRL even in toy-sized problems. Although more than a thousand deep RL repositories exist on Github, very few comes with adequate documentation or even a single learning curve, let alone systematic rigorous evaluations.

Many fields of AI, e.g., computer vision, have made significant progress powered by open-source software tools [97, 1, 158] and standardized benchmarks [182]. To sustain and facilitate the research of deep RL in robotics, we envision that it is vital to provide a flexible framework for rapid development of new algorithms and a standardized robotics benchmark for rigorous evaluation.

In this paper, we introduce the open-source framework SURREAL (**S**calable **R**obotic **RE**inforcement-learning **AL**gorithms). Standard approaches to accelerating deep RL training focus on parallelizing the gradient computation [140, 166]. SURREAL decomposes a distributed RL algorithm into four components: generation of experience (*actors*), storage of experience (*buffer*), updating parameters from experience (*learner*), and storage of parameters (*parameter server*). This decoupling of data generation and learning eliminates the need of global synchronization and improves scalability. SURREAL offers an umbrella support to distributed variants of both on-policy and off-policy RL algorithms. To enable scalable learning, we develop a four-layer computing infrastructure on which RL experiments can be easily orchestrated and managed. The system can be deployed effortlessly on commercial cloud providers or personal computers. Thanks to the layered design, our system can be fully replicated from scratch, which also contributes to the reproducibility of our experiments.

Furthermore, we introduce SURREAL Robotics Suite, a diverse set of physics engine-based robotic manipulation tasks, as an accessible benchmark for evaluating RL algorithms. Simulated systems have been traditionally used as a debugging tool in robotics to perform *mental rehearsal* prior to real-world execution [104]. A series of successful attempts have been made in utilizing simulated data for learning robot controllers [183, 265, 161, 92, 218]. We expect the simulation-reality gap to be further narrowed with more advanced simulation design and policy transfer techniques. We hope that this standardized benchmark, along with the open-source SURREAL codebase, will accelerate future research in closing the reality gap.

To this end, we develop well performing and distributed variants of PPO [194] and DDPG [126], called SURREAL-PPO and SURREAL-DDPG. We examine them in six of the Robotics Suite tasks with single-arm and bimanual robots. We report performance in various setups, including training on physical states or raw pixels, RL from scratch or aided by VR-based human demonstrations. We also quantify the scalability of distributed RL framework compared to popular open-source RL implementations [71, 45] in OpenAI Gym environments [15], the *de facto* standard continuous RL benchmark. The experiments show that SURREAL algorithms are able to achieve strong results and high scalability with increased numbers of parallel actors.

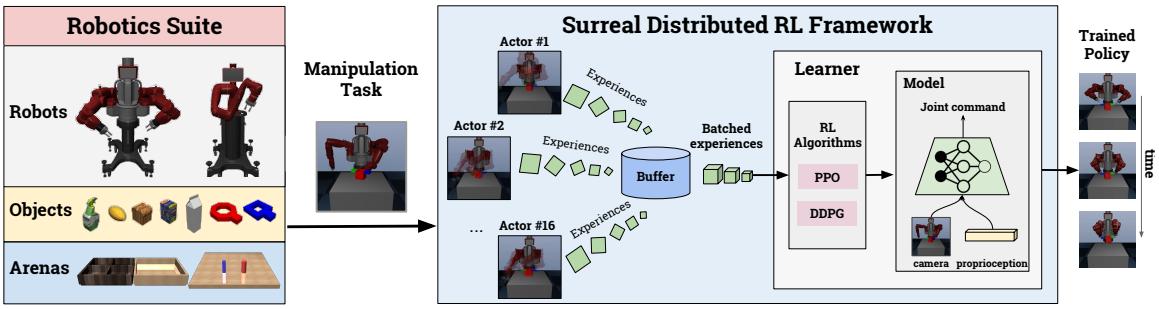


Figure 2.1: SURREAL is an open-source framework that facilitates reproducible deep reinforcement learning (RL) research for robot manipulation. We implement scalable reinforcement learning methods that can learn from parallel copies of physical simulation. We also develop Robotics Suite as an accessible benchmark for evaluating the RL agents' performances.

2.2 Related Work

2.2.1 Deep Reinforcement Learning in Robotics

Deep RL methods have been applied to mobile robot navigation [264, 157] and robot arm manipulation [119, 248, 25, 66, 165]. Both model-based and model-free RL approaches have been studied. Model-based methods [119, 248, 147] often enjoy sample efficiency of learning, but pose significant challenges of generalization due to their strong model assumptions. Model-free methods [265, 25, 66, 165] are more flexible, but usually require large quantity of data. In this work, we build a distributed learning framework to offer a unified support of two families of model-free continuous RL methods: value-based methods based on deterministic policy gradients [126] and trust-region methods [192, 78, 194]. Our focus on developing model-free RL methods and building simulated robotic benchmarks is encouraged by a series of recent progress on simulation-to-reality policy transfer techniques [183, 265, 161, 92, 218].

2.2.2 Distributed Deep RL Frameworks

As the learning community tackles problems of growing sizes and wider varieties, distributed learning systems have played an integral role in scaling up today's learning algorithms to unprecedented scales [197, 40]. The high sample complexity and exploration challenge in deep RL have accentuated the advantages of distributed RL frameworks. Prior approaches have relied on asynchronous SGD-style learning (e.g., A3C [140], Gorila [148], ADPG-R [166]), batched data collection for high GPU throughput (e.g., batched A2C [34], GA3C [9], BatchPPO [71]), and more recently, multiple CPU actors for experience generation and single GPU learner for model update (e.g., Ape-X [85] and IMPALA [49]).

SURREAL differs from asynchronous gradient methods like A3C, because the latter shares gradients between decentralized learners instead of sending experience, which is less desirable because gradients become outdated more rapidly than experience data. To date, Ape-X and IMPALA have reported state-of-the-art results with off-policy RL in several benchmarks. SURREAL resembles these two methods, which also separate experience generation from centralized learning. The major difference is that SURREAL provides a unified approach towards both on-policy trust region algorithms [194, 78] and off-policy value-based algorithms [126, 198].

2.2.3 Benchmarking Robotics and RL Research.

Both robotics and deep reinforcement learning communities have confronted with significant challenges of benchmarking and reproducibility. In robotics replicability and reproducibility has been a long-standing challenge [41, 131]. Attempts to improving this situation include annual robotic competitions [7, 37] and standardized object sets [19]. However, designing robotic benchmarks that can be both standardized and widely accessible remains to be an open problem. The deep RL community has been also faced with the same challenge of reproducibility [79], which hinders research progress and technology transfer.

2.3 Surreal Distributed Reinforcement Learning Framework

SURREAL's goal is to provide highly scalable implementations of distributed RL algorithms for continuous control. We develop distributed variants of the on-policy PPO [194] and off-policy DPG [126] algorithms, and unify them under a single algorithmic framework. We further develop a distributed computing infrastructure that can be easily replicated and deployed. Here we start with a brief review of the basics of the PPO and DPG algorithms.

2.3.1 Proximal Policy Gradient

Policy gradient algorithms [211] are among the most robust methods for continuous control. They aim to directly maximize the expected sum of rewards $J(\theta) = \mathbb{E}_{\tau_\theta}\left[\sum_t \gamma^{t-1} r(s_t, a_t)\right]$ with respect to the parameters θ of the stochastic policy $\pi_\theta(a|s)$. The expectation is taken over τ_θ, which denotes the trajectories induced by π_θ interacting with the environment. The vanilla policy gradient estimator is given by $\nabla_\theta J_{\text{PG}} = \mathbb{E}_{\tau_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) A_t\right]$. A_t is the advantage function, typically formulated as subtracting a value function baseline from the cumulative reward, $R_t - V(s_t)$.

Policy gradient estimates can have high variance (e.g. [47]). One effective remedy is to use a *trust region* to constrain the extent to which any update is allowed to change the policy. [163, 192, 236]. Trust Region Policy Optimization (TRPO; [192]) is one such approach that enforces a hard constraint on the Kullback-Leibler (KL) divergence between the old and new policies. It optimizes for a

surrogate loss as the following:

$$J_{\text{TRPO}}(\theta) = \mathbb{E}_{\tau_{\theta_{\text{old}}}}\left[\sum_t \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t\right]$$
$$\text{subject to } \text{KL}\left[\pi_{\theta_{\text{old}}}|\pi_\theta\right] < \delta$$

More recently, Proximal Policy Optimization (PPO) has been proposed as a simple and scalable approximation to TRPO. PPO only relies on first-order gradients and can be easily combined with recurrent neural networks (RNN) in a distributed setting. PPO implements an approximate trust region via a KL-divergence regularization term, the strength of which is adjusted dynamically depending on actual change in the policy in past iterations. PPO optimizes an alternative surrogate loss $J_{\text{PPO}}(\theta) = \mathbb{E}_{\tau_{\theta_{\text{old}}}}\left[\sum_t \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t - \lambda \cdot \text{KL}[\pi_{\text{old}}|\pi_\theta]\right]$, where λ is adjusted if the actual KL falls out of a target range. There are other flavors of PPO, but we only use the adaptive-KL version in this paper.

2.3.2 Deterministic Policy Gradient

Policies may also be induced by maximizing an estimate of expected cumulative reward [209, 210]. The action value function $Q_\pi(s,a) = \mathbb{E}\left[R_t|s,a\right]$ can be learned by the Bellman equation [211], which relates the value of a state-action pair (s,a) to the value of subsequent state-action pairs: $Q_\pi(s,a) = r + \gamma \mathbb{E}_{s',a'}\left[Q_\pi(s',a')\right]$.

Given a discrete action space, a greedy policy can be used to select the action associated with the highest Q value, which is performed by Deep Q Network (DQN) [142]. In continuous action space, however, it is difficult to find an analytical expression for the value-maximizing action. Actor-critic method solves the problem by training a policy, known as the "actor", to maximize the estimated expected return computed by a value function, known as the "critic". The deterministic policy gradient theorem [198] shows how to update the policy: $\nabla_\theta J_{\text{DPG}} = \mathbb{E}_{\tau_\theta}\left[\nabla_a Q_\pi(s,a)|_{a=\pi(s)} \nabla_\theta \pi_\theta(s)\right]$.

For a large state space like image observation, both the actor and the critic can be defined by deep neural networks. This variant is correspondingly called Deep Deterministic Policy Gradient (DDPG) [126]. Because DDPG is an off-policy method, we can use the experience replay technique to store past experiences and sample them in batches to update the deep neural networks.

2.3.3 Surreal Distributed RL Design

The distributed RL formulation in SURREAL consists of four major components illustrated in Fig. 2.2: actors, buffer, learner, and parameter server. Our key idea is to separate experience generation from learning. Parallel *actors* generate massive amount of experiences in the form of state-action transition tuples (s_t, a_t, s_{t+1}, r_t), while a centralized learner performs model updates. Each actor explores independently, which allows them to diversify the collectively-encountered state spaces.

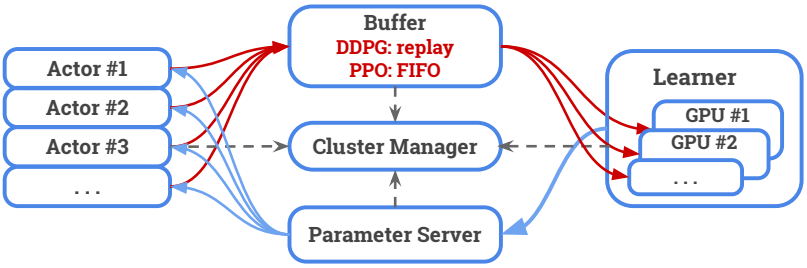


Figure 2.2: The SURREAL distributed components consists of *actors*, *buffer*, *learner*, and *parameter server*. The red arrows denote experience data flow and the blue arrows denote neural network parameter flow. All components report learning and system statistics to the *cluster manager*.

This alleviates the exploration challenge in long-horizon robotic manipulation tasks. The centralized learning eliminates global locking and reduces implementation complexity. Based on these design principles, we develop distributed versions of PPO and DDPG algorithms, which we will refer to as SURREAL-PPO and SURREAL-DDPG (pseudocode in Appendix).

On-policy and off-policy deep RL methods employ two different mechanisms of consuming experiences for learning. We introduce a centralized *buffer* structure to support both. In the on-policy case, the buffer is a FIFO queue that holds experience tuples in a sequential ordering and discards experiences right after the model updates. In the off-policy case, the buffer becomes a fixed-size replay memory [142] that uniformly samples batches of data upon request to allow experience reusing. The buffer can be sharded on multiple nodes to increase networking capacity.

The *learner* continuously pulls batches of experiences from the buffer and performs algorithm-specific parameter updates. Because learning is centralized, it can take advantage of multi-GPU parallelism. Periodically, the learner posts the latest parameters to the *parameter server*, which then broadcasts to all actors to update their behavior policies.

Due to our design's asynchronous nature and inevitable network latency, the actors' behavior policies that generate the experience trajectories can lag behind the learner's policy by several updates at the time of gradient computation. This would cause harmful off-policyness for on-policy methods. IMPALA [49] addresses this discrepancy by using a correction technique called V-trace. We propose a simpler alternative. SURREAL-PPO learner keeps a target network that is broadcasted to all actors at a lower frequency. This ensures that a much larger portion of the experience trajectories are on-policy, except for those generated within the policy lag (i.e. system delay between parameter server broadcasting target network parameters, to actors actually updating the behavior policy to the target network). Empirically, we find that the target network mechanism balances the trade-off between learning speed and algorithmic stability, which is crucial for agent performance.

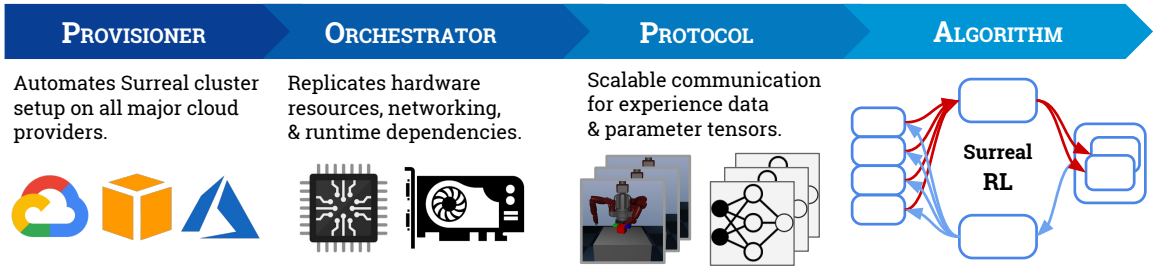


Figure 2.3: SURREAL reproducible and scalable learning infrastructure. The four layers from left to right are increasingly abstracted away from the hardware.

2.3.4 Surreal Heterogeneous Computing Infrastructure

Distributed RL, unlike data parallelism commonly used in supervised learning, requires complex communication patterns between heterogeneous components as seen in Fig. 2.2. This has increased the burden of engineering in distributed RL research. Our goal is to open source a set of well-engineered computing infrastructure that makes the runtime setup effortless to upper-level algorithm designers, with *reproducibility* and *scalability* as our guiding principles.

We design a four-layer distributed learning pipeline shown above, which decouples the RL algorithms from the underlying infrastructure (Fig. 2.3). SURREAL pipeline starts with the *provisioner* that guarantees the reproducibility of our cluster setup across Google Cloud, AWS, and Azure. The next layer, *orchestrator*, uses a well-established cloud API (Kubernetes) to allocate CPU/GPU resources and replicate the networking topology of our experiments. We use docker images to ensure that the runtime environment and dependencies can be exactly reproduced. Further down the pipeline, the *protocol* implements efficient communication directives. Some components can be sharded and load-balanced across multiple nodes to boost performance even further. We implement our *algorithm*s in PyTorch [158] with benefits of fast prototyping and dynamic computation graphs.

Among open-source distributed RL libraries, TensorFlow-Agent [71], OpenAI Baselines [45], and GA3C [9] provide very limited support for multi-node training, while SURREAL can easily scale to hundreds of CPUs and GPUs. Ray [146] is one of the systems that natively feature multi-node training. It has preliminary support for cloud, but only applies to AWS and does not automate cluster setup in a systematic manner as we do.

2.4 Robotics Suite: Simulated Robot Manipulation Benchmark

We aim to build a standardized and widely accessible benchmark with high-quality physical simulation, motivated by a series of recent work on leveraging simulated data for robot learning [183, 265,

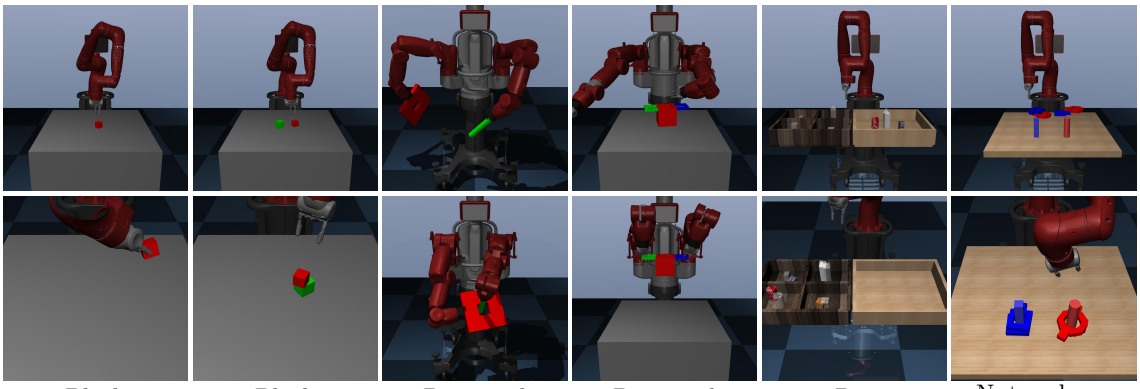


Figure 2.4: Six robot benchmarking environments. The first row shows initial configurations and the second row shows the states of task completion. When trained on raw pixel inputs, the agents take the RGB observations from the same cameras as illustrated in the second row.

92]. We develop the Robotics Suite in the MuJoCo physics engine [220], which simulates fast with multi-joint contact dynamics. It has been a favorable choice adopted by existing continuous control benchmarks [15, 213]. We provide OpenAI gym-style interfaces [15] in Python with detailed API documentations, along with tutorials on how to import new robots and create new environments and new tasks. We highlight four primary features in our suite:

1. **Procedural generation** (Fig. 2.1): we provide a modularized API to programmtically generate combinations of robot models, arenas, and parameterized 3D objects, enabling us to train policies with better robustness and generalization

2. **Control modes**: we support joint velocity controllers and position controllers to command the robots

3. **Multi-modal sensors**: we support heterogeneous types of sensory signals, including low-level physical states, RGB cameras, depth maps, and proprioception

4. **Teleoperation**: we support using 3D motion devices, such as VR controllers, to teleoperate the robots and collect human demonstrations.

Our current release of the benchmark consists of six manipulation tasks as illustrated in Fig. 2.4. All six environments are simulated at 10Hz control rate. The robots are controlled via joint velocity. There are three types of observations: proprioceptive features, object features, and camera observations. Proprioceptive features contain cos and sin of robot joint positions, robot joint velocities and current configuration of the gripper. Object features contain environment-specific values that describe the states and relationships of objects of interest. Camera observations are 84×84 RGB images.

We plan to keep expanding the benchmark with additional tasks, new robot models, and more advanced physics and graphics engines.

1. **Block Lifting:** A cube is placed on the tabletop. The Sawyer robot is rewarded for lifting the cube with a parallel-jaw gripper. We randomize the size and the placement of the cube.
2. **Block Stacking:** A red cube and a green cube are placed on the tabletop. The Sawyer robot is rewarded for lifting the red cube with a parallel-jaw gripper and stack it on top of the green cube.
3. **Bimanual Peg-in-hole:** The Baxter robot holds a board with a squared hole in the center in its right hand, and a long stick in the left hand. The goal is to move both arms to insert the peg into the hole.
4. **Bimanual Lifting:** A pot with two handles is placed on the tabletop. The Baxter robot is rewarded for lifting the pot above the table by a threshold while not tilting the pot over 30 degrees. Thus the robot has to coordinate its two hands to grasp the handles and balance the pot.
5. **Bin Picking:** The Sawyer robot tackles a pick-and-place task, where the goal is to pick four objects from each category in a bin and to place them into their corresponding containers.
6. **Nut-and-peg Assembly:** Two colored pegs are mounted to the tabletop. The Sawyer robot needs to declutter the nuts lying on top of each other and assembles them onto their corresponding pegs.

Our rationale of designing these tasks is to offer single-arm and bimanual manipulation tasks of large diversity and varying complexity. The complexity of a task is measured by the estimated number of steps an optimal agent can solve it and from the mean and variance of the time required by an experienced human operator using teleoperation routines that interface with a virtual reality controller. These mean episode durations from the successful human demonstrations — taken to be a proxy for task difficulty — are shown in Fig. 2.5. We infer that the Nut-and-peg Assembly and Bin Picking tasks are the hardest, whereas the Block Lifting and Block Stacking tasks are relatively easier. These human demonstrations were also used by our RL algorithms to accelerate exploration, as explained in Sec. 2.5.3. In the next section, we provide quantitative evaluations of SURREAL algorithms on these benchmarking tasks.

2.5 Experiments

We evaluate our SURREAL distributed RL algorithms in all six benchmarking tasks introduced in Sec. 2.4. For each task, we train RL agents with SURREAL-PPO and SURREAL-DDPG algorithms

on two settings: ground-truth physical states and raw pixel observations. In the former case, we use low-dimensional object features (object positions, rotations, etc.) and proprioceptive features (robot joint positions and velocities) as input. In the latter case, we use RGB camera observations and proprioceptive features, which are usually available on real robots.

In next two sections, we provide more details on the training algorithm setup.

2.5.1 Surreal-PPO Details

When training with pixel inputs, an $84 \times 84 \times 3$ RGB image is fed into a convolutional neural network (CNN) feature extractor. The extractor consists of an 8×8 convolution with 16 filters and stride 4, followed by a 4×4 convolution with 32 filters and stride 2, with ReLU activations. The convolution outputs are flattened and passed into a linear layer of size 256, which is concatenated to the proprioceptive inputs. The concatenated feature is fed into a 1-layer long short-term memory (LSTM) network with cell size 100. The LSTM output is fed into two separate feedforward networks for actor and critic. Both feedforward networks have hidden layers of size 300 and 200. The actor network outputs an action mean and log of standard deviation for each action dimension, whereas the critic network outputs a scalar. The actions are sampled with action mean and standard deviation kept by the actor network before feeding back to the environment.

To compute the loss, we first compute advantage for each timestep using generalized advantage estimate (GAE) [193]. We find the adaptive KL variant of PPO [194] to be more stable. In the distributed setting, when the learner publishes its parameters, agents may still be doing rollouts using an outdated set of parameters. This communication latency makes the algorithm not strictly on-policy, leading to divergent behaviors. In SURREAL-PPO learner, a set of target network parameters is kept along side of a set of updated model parameters. The target model parameters are broadcasted to the actors at a lower frequency to stabilize learning.

We use the number of model update steps before broadcasting to measure the target network update frequency: more model update steps correspond to a lower frequency. One way to interpret the reference target model is that it maintains a fixed center of the trust region. The learner is encouraged to optimize policy within the trust region since the adaptive KL-penalty variant of PPO penalizes excess KL-divergence. We also find that normalizing the low-dimensional states with a running estimate of mean and variance (z-filtering) helps stability and convergence for SURREAL-PPO.

2.5.2 Surreal-DDPG Details

When training with pixels, we stack the most recent 3 camera observations as input to the DDPG model. The stacked images are then fed through an 8×8 convolution with 16 filters and stride 4, followed by a 4×4 convolution with 32 filters and stride 2, with ReLU activations. The convolution

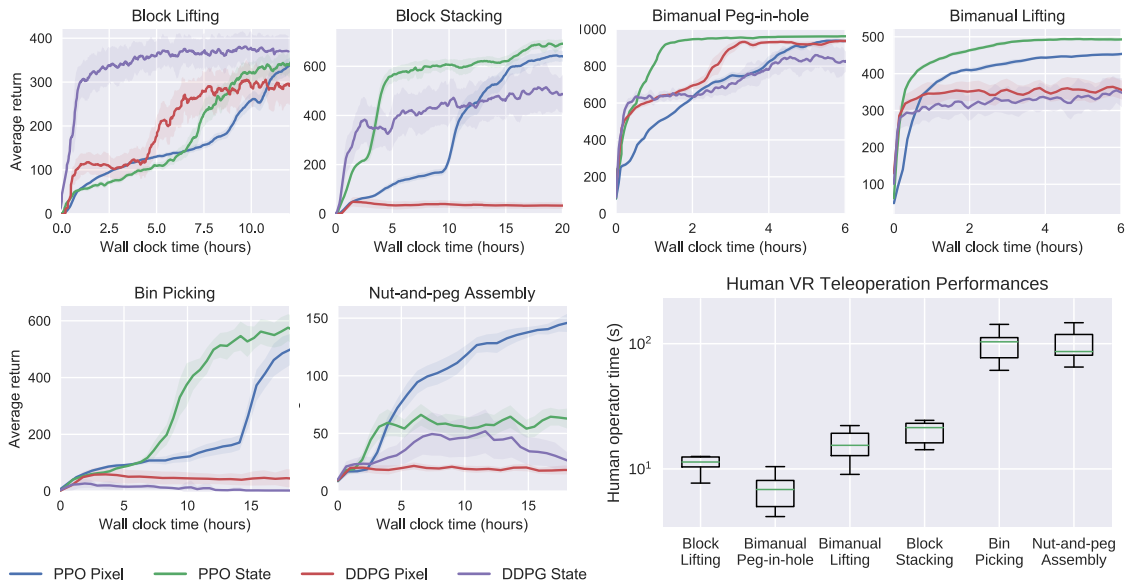


Figure 2.5: We report training curves of our SURREAL-DDPG and SURREAL-PPO on six SURREAL Robotics Suites tasks. The training curve represents the mean with standard deviation as the translucent band. We train the agents on both ground-truth physical states and raw pixel observations. We measure the complexity of the tasks as the median completion time by an experienced human operator using VR controllers to teleoperate the robots using APIs in Robotics Suite.

filter parameters are shared between the actor and critic networks. For the purpose of gradient updates, the convolution parameters are updated only by gradient descent on the critic loss.

Each actor explores using Ornstein-Uhlenbeck noise. Each actor is assigned an exploration noise σ parameter that remains constant over the course of training, where σ is scaled linearly between a minimum of 0 and a maximum of 1.0 across the actors.

For the Robotics Suite experiments, we find them to be more sensitive to layer normalization than the Gym environments. Bimanual Peg-in-hole experiments are run with layer normalization, 6-step rewards, no weight decay, and a maximum σ value of 1.0. Block lifting experiments use no layer normalization, 3-step rewards, weight decay of 0.0001, and a maximum σ value of 2.0.

2.5.3 Performances: Robotics Suite

Fig. 2.5 shows learning curves of our SURREAL-DDPG and SURREAL-PPO implementations on six Robotics Suite environments. We notice that SURREAL-PPO is able to converge to policies with lower standard deviations than SURREAL-DDPG. This is because SURREAL-PPO adjusts its exploration noise standard deviation throughout training, whereas SURREAL-DDPG uses fixed exploration noise. This is also reflected by the fact that the mean reward of SURREAL-PPO varies

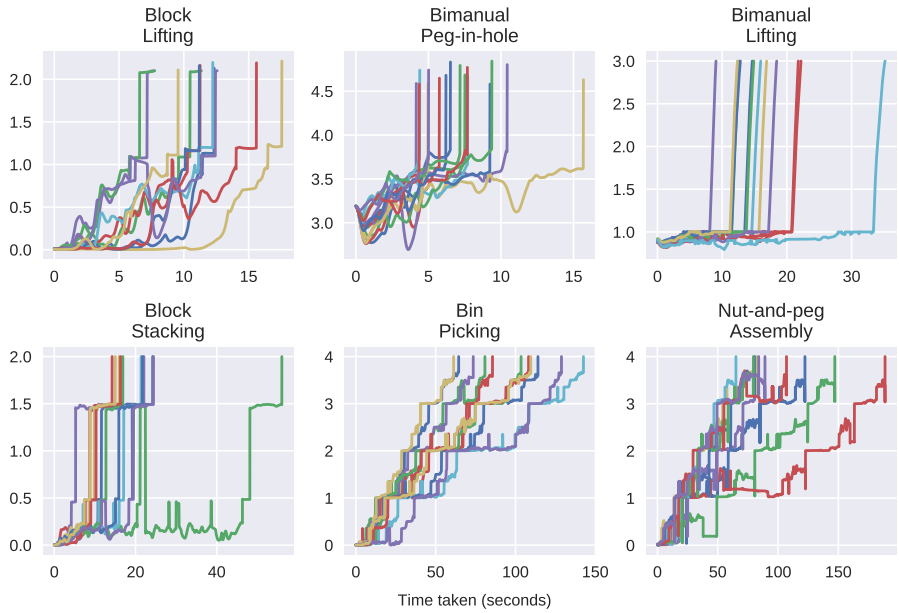


Figure 2.6: Sample immediate reward trajectories for successful task completions, collected using robot teleoperation facilities.

more smoothly than that of SURREAL-DDPG.

Block Lifting and Bimanual Peg-in-hole are the easiest tasks that consist of a single stage. Both algorithms can solve the tasks with policies trained on both input modalities. We notice that training time for Block Lifting is longer than Bimanual Peg-in-hole even though these two tasks are similar in complexity measured by median human completion time illustrated in Fig. 2.5. We hypothesize that this is caused by random initialization of object size in addition to position and orientation. Thus, convergence of training indicates that our algorithms can find robust solutions capable of adapting to environment variations.

The tasks of Block Stacking (grasping, lifting, and stacking) and Bimanual Lifting (grasping handles and lifting the pot) have longer horizons. Our algorithms can solve subtasks and achieve intermediate rewards: in the Bimanual Lifting task, actors learn to place the grippers on the handles but not to lift. We hypothesize that we need better exploration strategies to solve these tasks. Following [265], we build a curriculum from human demonstration to assist exploration. With some probability α, we initialize episodes with states taken along successful trajectories from the demonstrations (Fig. 2.6).

We experiment with three different curricula: in the "uniform" case the state is chosen uniformly at random from the entire demonstration dataset; "forward" samples states from the beginning of demonstration trajectories with a slowly growing window; "reverse" samples from the end of trajectories instead. Fig. 2.7 shows training results for Bimanual Lifting (states) with these curricula,

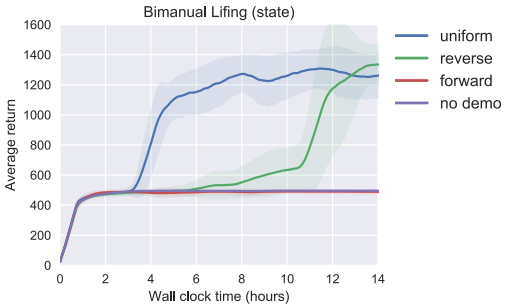


Figure 2.7: PPO agent trained on state for the Bimanual Lifting task with different types of demonstration curricula.

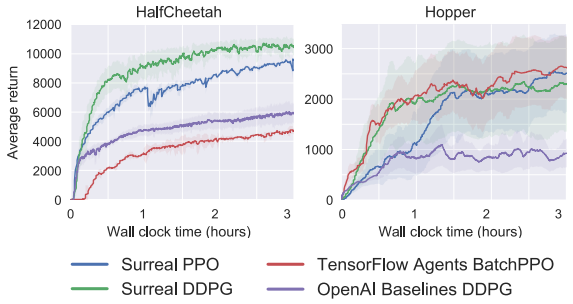


Figure 2.8: Training curves with 16 actors: our SURREAL algorithms and baselines on OpenAI Gym HalfCheetah and Hopper environments.

using 64 actors. We see that with proper strategies, SURREAL-PPO is able to complete the full task, which is previously only partially solved.

Bin Picking and Nut-and-peg Assembly are intrinsically more difficult because they have long horizons and multiple stages. As indicated in Fig. 2.5, they take the longest time to complete for humans. RL agents are able to perform a specific subtask but unable to proceed. In the Bin Picking task, for example, the PPO agents are able to successfully pick up, move, and drop at most one out of the four items. We believe solving these tasks is beyond the scope of our current algorithms.

2.5.4 Performances: OpenAI Gym

To put the performances of our distributed RL implementations in context, we run SURREAL on *de facto* continuous RL benchmark environments used in previous work [79, 47]. We compare with OpenAI Baselines DDPG [45] and TensorFlow Agent BatchPPO [71], which are among the popular open-source distributed reference implementations. Fig. 2.8a compares the learning curves of our DDPG and PPO implementations against the baselines on wall-clock time. All algorithms are trained with 16 actors with the same hardware allocation. Except for OpenAI-DDPG that does not support GPU out of the box, all other experiments use a single Nvidia P100 GPU for the learner.

Our algorithms outperform all baselines on `HalfCheetah` by a large margin and perform on a comparable level as BatchPPO on `Hopper`. We hypothesize that the differences in both algorithmic and system design contribute to the performance gap. BatchPPO distribute the experience generation by producing a batch of actions synchronously, which would be bottlenecked by the slowest simulation. OpenAI-DDPG collects gradients in a synchronous fashion, which is not as desirable as communicating experiences asynchronously (see Sec. 2.3.3).

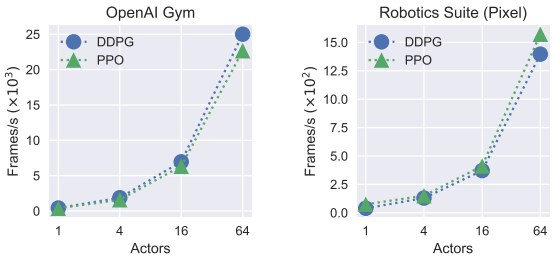


Figure 2.9: Total actor throughput: environment FPS from all actors combined scales linearly with the number of actors in both algorithms.

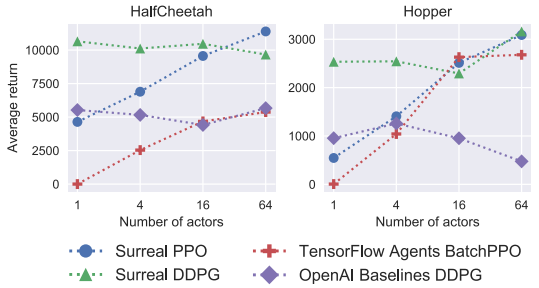


Figure 2.10: Scalability with respect to the number of actors: our methods v.s. baselines on gym HalfCheetah and Hopper environments.

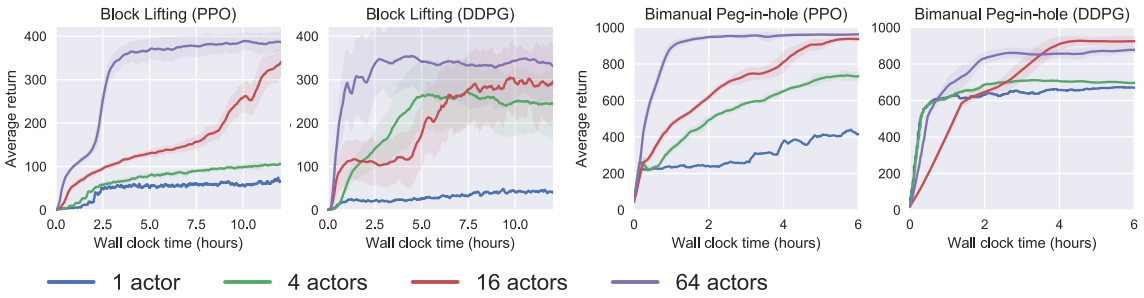


Figure 2.11: Learning curves for SURREAL-PPO and SURREAL-DDPG trained on the block lifting and bimanual peg-in-hole tasks with raw pixel inputs using different number of actors, ranging from 1, 4, 16, 64. Both algorithms learn sizeably faster with more actor experience throughput.

2.5.5 Scalability

Fig. 2.9 shows our system characteristics on Robotics Suite (training on pixels) and Gym environments (training on states). The total actor throughput is number of environment frames collected by all actors per second. Our Buffer is sharded and load-balanced to reduce network congestion, which yields an almost linear speedup with respect to the number of actors. The scalability of our algorithms becomes more remarkable in our tasks, where complex dynamics and graphical rendering makes simulation slower than OpenAI Gym by an order of magnitude .

We evaluate the scalability of our methods with respect to varying numbers of actors. Fig. 2.10 shows episode rewards on Gym `HalfCheetah` and `Hopper` environments. Both our methods and the baselines are trained for 3 hours using 1, 4, 16 and 64 actors with the same hardware resource allocation. Our methods generally obtain better performance with growing number of actors, and score higher than the baselines across all actor settings. SURREAL-PPO is on-policy and uses every experience once; its learner speed is bound by total actor throughput until the learner machine

capacity is saturated. Thus increasing the number of actors speeds up the learner, as demonstrated by the monotonically increasing curves in Fig. 2.10. In contrast, the DDPG learner samples from a replay buffer and reuses experiences; its speed is not directly correlated with the total actor throughput. If a small number of actors is enough to explore the environment extensively, the learner can still learn at the maximal possible speed. This is the case for Gym environments where simulation is fast. SURREAL-DDPG achieves the maximum score with as few as 1 or 4 actors.

In comparison, on Robotics tasks with slower simulation, a large number of actors is needed for both SURREAL-PPO and SURREAL-DDPG to learn; they show better performance when trained with more actors. Fig. 2.11 compares the learning curves of our DDPG and PPO on Block Lifting and Bimanual Peg-in-hole with pixel inputs. Training with 64 actors results in faster learning and better final performance. We hypothesize that, given fewer actors (especially 1 and 4 actors), both DDPG and PPO suffer from inadequate exploration, which causes them to learn much slower or become trapped at sub-optimal policies.

2.6 Conclusion

We address the challenge of reproducibility and benchmarking in deep reinforcement learning and robot manipulation research. We introduce SURREAL, a highly-scalable distributed framework that supports on-policy and off-policy deep RL algorithms. We develop a novel system infrastructure to enable reproducibility and extensibility. To rigorously evaluate the performance of the RL algorithms, we introduce Robotics Suite, which contains a set of manipulation tasks with varying levels of complexity. We illustrate that our distributed RL algorithms outperform widely used RL libraries on standard benchmarks, and perform well in manipulation tasks in our new benchmark.

In the future, we plan to expand SURREAL with new algorithms and enrich the benchmark with new tasks. We hope SURREAL will become a valuable resource for a broad range of manipulation related research, as a training resource, a standardized benchmark, and a framework for rapid algorithm development.

Chapter 3

System Design for Massively Parallel Reinforcement Learning

3.1 Introduction

Distributed systems development is becoming increasingly important in the field of deep learning. Prior work has demonstrated the value of distributing computation to train deep networks with millions of parameters on large, diverse datasets [40, 62, 146]. Distributed learning systems have recently witnessed a great deal of success across a wide variety of games, tasks, and domains [40, 197, 62, 124, 85, 49]. In particular, distributed Reinforcement Learning (RL) has demonstrated impressive state-of-the-art results across several sequential decision making problems such as video games and continuous control tasks.

Three important design paradigms are desired for distributed reinforcement learning systems: reproducibility, flexibility, and scalability. Reproducing and validating prior deep reinforcement learning results is rarely straightforward, as it can be affected by numerous factors in the tasks and the underlying hardware. It makes comparing and evaluating different algorithms difficult [79]. In order to ensure progress in the field, a distributed reinforcement learning system must produce results that are easily *reproducible* by others. Furthermore, to support a wide spectrum of algorithms and enable the creation of new algorithms, the distributed system must also be *flexible*. Implementing a new RL algorithm should not require re-engineering intermediate system components or adding new ones. Instead, existing system components need to be able to adapt to the needs of upstream algorithms and use cases. Finally, the system should exhibit significant capability to *scale* to leverage large quantities of physical computing resources efficiently. Deep RL methods often suffer from high sample complexity due to the burden of exploring large state spaces and can benefit from diverse sets of experience [85]. Leveraging large-scale distributed computation can help these methods collect

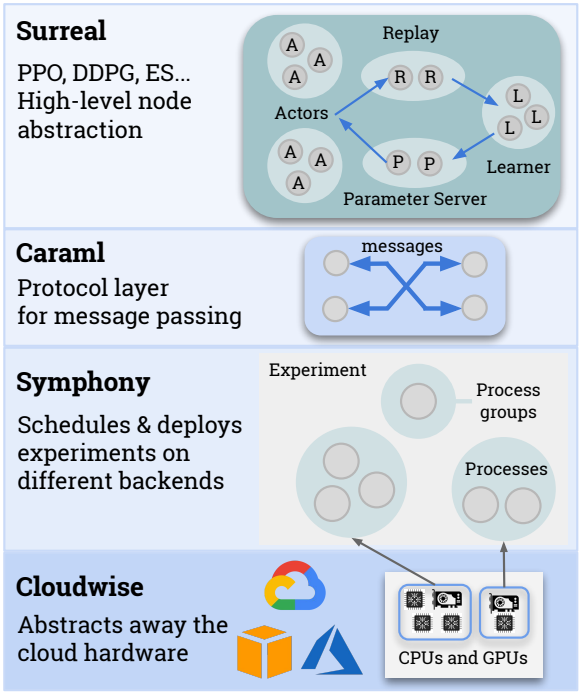


Figure 3.1: Overview of the four-layer stack in SURREAL-SYSTEM: CLOUDWISE for cloud provisioning, SYMPHONY for container orchestration, CARAML as a communication layer, and SURREAL for distributed reinforcement learning algorithms.

diverse experience quickly and help improve convergence rates.

Although existing distributed reinforcement learning algorithms such as Ape-X [85] and IMPALA [49] demonstrate an impressive capacity to scale up to many machines and achieve state-of-the-art results across a wide spectrum of Atari benchmarks and continuous control domains, these results are not easily reproducible by other researchers due to the difficulty of reimplementing the software infrastructure and gathering the hardware resources necessary to recreate their experiments. Many existing open-source reinforcement learning frameworks aim at providing high-quality implementations of these standard distributed RL algorithms, but they do not provide fully-integrated support for the corresponding hardware resources [45, 71, 124].

Other existing systems in deep learning have also addressed some of these design paradigms. TVM [29] is an open-source end-to-end optimizing compiler for maintaining deep learning model performance across different hardware backends, while VTA [145] builds on top of the TVM compiler to provide an entire deep learning stack that allows for control over both high-level software and low-level hardware optimizations. While neither of these frameworks support scalability, the end-to-end control of both hardware and software allows for both reproducibility of model performance and

a flexible approach to designing models that can leverage different types of hardware capabilities.

In order to establish a distributed RL framework that produces consistent and reproducible results that can be verified and built upon by researchers, the framework must provide an end-to-end solution, from hardware provisioning and deployment to algorithms that operate at the highest level of abstraction. Enabling control over hardware resources can benefit the flexibility and scalability of such a system as well by supporting algorithms that might require several computing nodes running on heterogenous hardware and software specifications. Therefore, we propose a general open-source sequential decision making framework that provides the entire stack — from hardware deployment to algorithms.

We present SURREAL-SYSTEM, a fully-integrated stack for distributed deep reinforcement learning. SURREAL-SYSTEM consists of four primary layers. The first layer, the Provisioner (CLOUDWISE library), offers a common hardware abstraction for instance types and node pools across different cloud vendors. It prepares the foundation for the next layer, the Orchestrator (SYMPHONY library), that builds upon a community-standard cloud API (Kubernetes). SYMPHONY allows users to specify an experiment's launch logic, hardware resources, and network connectivity patterns in high-level description. It provides a unified frontend for orchestration and deployment to different backends ranging from local laptop to cloud clusters. Once an experiment has been configured, the Protocol layer (CARAML library) handles all the communications between algorithmic components. Simulated experience data and neural network parameters can be transferred very efficiently, thanks to the RL-specific optimizations we make. Finally, we provide competitive implementations of policy gradient algorithms and Evolution Strategies (SURREAL libary) as well as demonstrating their performance and capacity to scale in large-scale experiments.

In Sec. 3.5 we show how our stack scales efficiently to heterogeneous clusters having 1000s of CPUs and 100s of GPUs when using both Proximal Policy Optimization(PPO) and Evolution Strategies (ES). We achieve near linear scaling in environment frames per second with number of agents demonstrating the effectiveness of our communication stack. We also show how optimizations like a load balanced replay, batched actors and optimized communication help us further improve system performance.

We also show the learning performance of our algorithms in Sec. 3.6 for a range of OpenAI Gym tasks as well as more challenging Robotics Suite tasks introduced in our prior work [208]. By using 1024 agents, our PPO implementation is able to partially solve even the most challenging tasks which were unsolved when using fewer agents hence showing the advantage of massively distributed RL. Our ES implementation also outperforms the reference implementation [124] on a range of OpenAI Gym tasks. The contributions of our work are as follows:

1. We propose SURREAL-SYSTEM, a reproducible, flexible, and scalable framework for distributed reinforcement learning.

2. We introduce a streamlined four-layer stack that provides an end-to-end solution from hardware

to algorithms. It can both massively scale on full-fledged cloud clusters and quickly iterate research ideas on a local machine.

3. We describe the details and perform evaluations of our algorithmic implementations on a variety of control tasks. They show highly competitive with the state-of-the-art results.

3.2 Related Work

Reinforcement learning methods, powered by deep neural networks, often require a significant amount of experience for learning. This has accentuated the advantages of large-scale distributed learning methods for learning efficiency. A series of deep learning models and algorithms have been proposed for large-scale sequential decision making. One notable class of architectures and algorithms scales deep RL methods by using asynchronous SGD to combine gradients and update global parameters. The *Gorila* architecture [148] proposes to use multiple actors, multiple learners, a distributed experience replay memory, and a parameter server that aggregates and applies gradients shared by the learners. A3C [140] instead utilizes several CPU threads on a single machine, where each thread is an actor-learner that shares gradients with the other threads. A distributed version of PPO was also introduced by [78] with several workers that collect experience and send gradients to a centralized learner. However, sharing gradients instead of experience is less desirable since gradients become stale and outdated much more quickly, especially for off-policy methods.

Another class of architectures scales deep RL methods via several actors that solely collect and send experience, a distributed replay memory and parameter server, and one centralized learner that performs parameter updates using experience sampled from the replay memory. Ape-X [85], IMPALA [49], and SURREAL-PPO implementation fall into this category. In contrast to prior work which focused on off-policy methods, we extend this paradigm to accommodate on-policy learning.

Prior work has also shown the efficacy of distributed algorithms based on evolutionary computation. [186] applied Evolution Strategies (ES) to common deep RL benchmarks and found that their method could scale to many more distributed workers than RL algorithms while achieving competitive results. Similarly, [207] found that a simple genetic algorithm (GA) could leverage distributed computation much more effectively than RL methods, which suffer from bottlenecks during learning, and also produce competitive results on several domains.

Existing frameworks of distributed RL algorithms exploit both data parallelism and model parallelism to learn on massive data [40]. Open-source libraries for distributed RL include OpenAI Baselines [45], TensorFlow Agents [71], Neuroevolution [207], etc. These libraries focus on algorithm implementations built on third-party learning frameworks, such as Tensorflow and PyTorch, without providing infrastructure support for computing runtime. Ray RLlib [124] is built on the Ray distributed framework designed for machine learning applications. It also has flexibility in supporting different types of distributed algorithms, including Evolution Strategies (ES) and Proximal

Table 3.1: SYMPHONY backend feature comparisons

Backends	Scalability	Autoscaling	Containerized	Debugging
Kubernetes	Multi-node on cloud	Yes	Yes	Slow
Tmux	Local only	No	No	Fast
Docker-compose	Multi-node only with `Swarm`	No	Yes	Fast
Minikube	Local only, simulates cloud	No	Yes	Slow

Policy Optimization (PPO). In contrast to Ray RLlib, SURREAL also provides a set of toolkits that sit between the learning algorithms and the hardware for scalable deployment in different computing platforms.

3.3 Distributed Reinforcement Learning Full Stack

Unlike data parallelism in supervised learning, distributed reinforcement learning (RL) algorithms require complex communication patterns between heterogeneous components, increasing the burden of engineering in reinforcement learning research. Our goal is to open source a computing infrastructure that makes the runtime setup effortless to upper-level algorithm designers with *flexibility*, *reproducibility* and *scalability* as our guiding principles.

We design a four-layer distributed learning stack as shown in Fig. 3.1, which decouples the end-user algorithms from the underlying computing runtime. Users of SURREAL-SYSTEM should be able to replicate our cluster setup, reproduce our experimental results, and rapidly iterate new research ideas on top of our algorithmic libraries. Each component can be used independent of each other, or even outside the SURREAL context to facilitate other cloud-based distributed tasks. Here we provide an overview of our four-layer stack for distributed RL systems, from the cloud computing hardware to the RL algorithm implementations.

3.3.1 Provisioner: Cloudwise

CLOUDWISE aims to achieve a fully reproducible SURREAL cluster setup running on the users' cloud account. CLOUDWISE abstracts away the cloud instances and node-pool configurations between different cloud providers. For example, Google Cloud machine type identifiers look like `n1-standard-4` and `n1-highmem-64`, while Amazon AWS uses a distinct naming scheme such as `t2.small` and `m5d.24xlarge`. They also have different mechanisms to deploy native clusters.

To standardize across different conventions, CLOUDWISE makes the following two design choices:

1. Lifts a cloud account to be Kubernetes-ready. Kubernetes [17] is a well-established, open-source cloud API standard compatible with all major cloud providers.

2. Uses descriptive Python parameters (e.g. `cpu=48, gpu_type="v100", gpu=4`) that translate to the corresponding terminology on different cloud services.

CLOUDWISE automates the tedious and convoluted procedure of setting up a customized SURREAL Kubernetes cluster from scratch. The library only needs to be run once before any experiment. After the setup, the upper layers will not be able to tell apart the differences between cloud providers, such as GCE or AWS. CLOUDWISE is also capable of adding, removing, and editing the properties of node pools after the cluster has been created.

3.3.2 Orchestrator: Symphony

Once the user sets up the cloud account with CLOUDWISE, the SYMPHONY library takes over and helps orchestrate `Experiment`s on top of Kubernetes.

Each `Experiment` is a logical set of `Process`es and `ProcessGroup`s that communicate with each other through intra-cluster networking. A `ProcessGroup` contains a number of `Process`es that are guaranteed to be scheduled on the same physical node. Users can easily specify per-process resources (e.g., CPU-only node for actors and GPU nodes for learner) as well as their network connectivity, while SYMPHONY takes care of scheduling and book-keeping. It uses containerization (Docker) to ensure that the runtime environment and dependencies are reproducible.

Furthermore, SYMPHONY supports auto-scaling out of the box. The auto-scaling mechanism spins up or tears down nodes as new experiments start or old experiments terminate. It is highly economical for small research groups because no cloud instances are left running without a workload. Team members can collaborate on the same cluster and check each others' experiment status.

Multiple Orchestration Backends

A prominent feature of SYMPHONY is its flexibility to deploy the same experiment logic on different computing environments, ranging from a personal laptop to a full-fledged cloud cluster. In this regard, SYMPHONY provides the users with a unified interface for process orchestration and networking. Once users specify an abstract `Experiment` and connectivity configuration, they can choose from a variety of backends that satisfy different stage of development. We currently support the following four modes (also summarized in Table 3.1):

Kubernetes ("Kube") backend. This is the primary use case of SYMPHONY. It deploys to the Kubernetes engine on the users' cloud provider, and its scalability is limited only by the resource quota and budget. All processes ("pods" in Kubernetes terminology) are containerized, thus avoiding dependency issues. Kubernetes is suitable for deploying large-scale experiments after validating on a smaller scale. To enable fast iteration and development, we provide Tmux, Docker-compose and Minikube as well.

Tmux backend. `tmux` is a terminal multiplexer that allows users to access multiple terminal sessions in separate "panes" inside a single window. We build the backend upon `tmux`'s panes to

deploy a distributed experiment on an interactive machine (personal laptop/desktop or ssh-reachable machine). The merit is that we can now quickly iterate on the codebase. Any code updates will be reflected immediately in the local `tmux` session; error messages, if any, will emerge almost instantly. The demerit is that there is no containerization, and users will be responsible for installing the dependencies manually. In addition, `tmux` does not support multi-node distributed training.

Docker-compose backend. This backend is a reconciliation of Tmux and Kube modes. It deploys locally on any interactive machine that has `docker` installed. `Docker-compose` eliminates the need to install complex dependencies locally and does not incur the Kube pod creation overhead. In future upgrades, we will also support `docker-swarm`, which will enable multi-node communication for this backend.

Minikube backend. Conceptually the same as the Kube backend, `minikube` deploys on a local machine and simulates the full-blown cloud environment. Users can test their implementation thoroughly before running on the cloud account to avoid any unnecessary computing costs. Minikube backend has exactly the same API and command line interface as Kube backend, thus the migration to the real cluster will be relatively effortless.

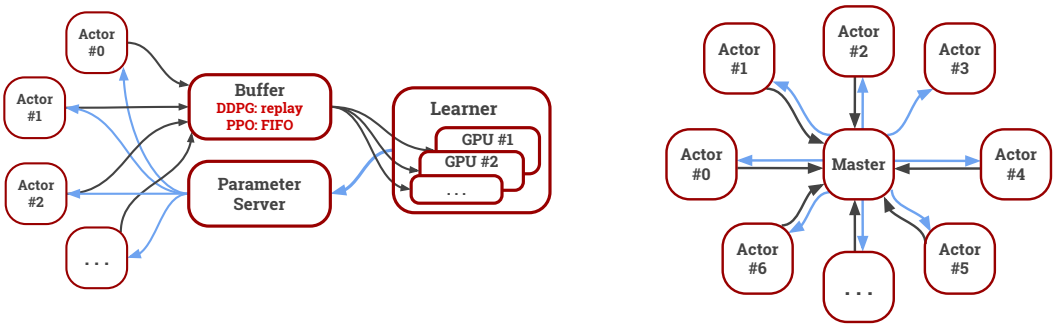
Load-balancing and Sharding

Kubernetes is designed to provide horizontal scalability and handle large workloads. When using Symphony and Caraml on Kubernetes, components of a distributed learning algorithm can be sharded and requests to them are load balanced. For example, when a single replay server is not able to handle data generated by a large number of actors, one can create multiple shards. Symphony manages service declaration on Kubernetes to ensure that they process workload evenly. The effect of sharding the replay server is further discussed in Sec. 3.5.3.

3.3.3 Protocol: Caraml

The Caraml library (*CAR*efree *A*ccelerated *M*essaging *L*ibrary) is our communication protocol based on ZeroMQ (a high performance messaging library) and Apache Arrow (in-memory data format for fast serialization). Caraml implements highly scalable distributed directives, such as `Push-Pull` for actors sending experience to replay and `Publish-Subscribe` for broadcasting parameters to actors. Caraml offers a more transparent alternative to frameworks like distributed TensorFlow [1], and applies to use cases beyond machine learning as well.

In distributed learning scenarios, high-dimensional observation vectors and large network parameters frequently make communication and serialization the bottleneck of the system. Caraml provides specific optimizations to avoid these problems. For example, we use Caraml Data Fetcher to fetch data from separate processes (actors) and transfer to the main process using a shared memory



(a) Distributed System Diagram for Reinforcement Learning

(b) Distributed System Diagram for Evolution Strategies

Figure 3.2: Our framework provides abstractions for composing heterogeneous parallel components for different distributed learning algorithms. In this work, we demonstrate two types of algorithms: a) System diagram for on-policy and off-policy reinforcement learning methods. Actor nodes share their experiences with a buffer server, that relays them to the learner server. The updated parameters are broadcast back to the actors via the parameter server. b) System diagram for Evolution Strategies implementation. The actors communicate their experiences to a master node to be broadcasted to all nodes. Parameter updates happen simultaneously in all actors.

(reply buffer). This offloads serialization and networking from the main process. We quantitatively examine the speed-up of RL algorithms from CARAML's optimizations in Sec. 3.5.5.

3.3.4 Algorithm: Surreal

The algorithm layer at the top of hierarchy uses high-level abstractions of orchestration or communication mechanisms. The clean decoupling between algorithms and infrastructure fulfills our promise to deliver a researcher-friendly and performant open-source framework.

SURREAL provides a flexible framework for composing heterogenous parallel components into various types of distributed RL methods. It allows the end users to rapidly develop new algorithms while encapsulating the underlying parallelism. To showcase the flexibility, we develop two distributed learning algorithms with contrastive patterns of parallelism as illustrated in Fig. 3.2: Proximal Policy Optimization (PPO) and Evolution Strategies (ES). PPO requires the coordination of different types of components, including *actors*, *learner*, *buffer*, and *parameter server*. In contrast, ES is easily parallelizable with an array of homogeneous actors and a master node for model update. We provide an overview of the PPO and ES algorithms in Sec. 3.3.4 and Sec. 3.3.4 respectively. We perform system analysis and quantitative evaluation on our implementations in Sec. 3.5 and Sec. 3.6.

In addition to SURREAL-PPO and SURREAL-ES, our prior work [208] has also implemented a variant of Deep Deterministic Policy Gradient (DDPG) [126] in the SURREAL framework. SURREAL-DDPG uses the same setup as SURREAL-PPO, involving actor, learner, replay and parameter server.

In contrast to PPO, DDPG is off-policy and reuses observation. Its replay servers thus contain a big replay memory, entries in which are sampled by the learner. From our experiments, we have seen a better learning scalability of SURREAL-PPO, so we focus on discussing the learning performances of SURREAL-PPO. A more complete comparisons between these two algorithms can be found in [208]. SURREAL-DDPG will be relevant to us in Sec. 3.5 where we discuss several design choices to build a system that can accommodate both PPO and DDPG algorithms in a unified abstraction.

Proximal Policy Optimization

Policy gradient algorithms [211] are among the most robust methods for continuous control. They aim to directly maximize the expected sum of rewards $J(\theta) = \mathbb{E}_{\tau_\theta}\left[\sum_t \gamma^{t-1} r(s_t, a_t)\right]$ with respect to the parameters θ of the stochastic policy $\pi_\theta(a|s)$. The expectation is taken over τ_θ, which denotes the trajectories induced by π_θ interacting with the environment. The vanilla policy gradient estimator is given by $\nabla_\theta J_{\text{PG}} = \mathbb{E}_{\tau_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) A_t\right]$. A_t is the advantage function, typically formulated as subtracting a value function baseline from the cumulative reward, $R_t - V(s_t)$.

Policy gradient estimates can have high variance. One effective remedy is to use a *trust region* to constrain the extent to which any update is allowed to change the policy. Trust Region Policy Optimization (TRPO) [192] is one such approach that enforces a hard constraint on the Kullback-Leibler (KL) divergence between the old and new policies. It optimizes a surrogate loss $J_{\text{TRPO}}(\theta) = \mathbb{E}_{\tau_{\theta_{\text{old}}}}\left[\sum_t \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t\right]$ subject to $\text{KL}\left[\pi_{\theta_{\text{old}}}|\pi_\theta\right] < \delta$. More recently, Proximal Policy Optimization (PPO) has been proposed as a simple and scalable approximation to TRPO [194]. PPO only relies on first-order gradients and can be easily combined with recurrent neural networks (RNN) in a distributed setting. PPO implements an approximate trust region via a KL-divergence regularization term, the strength of which is adjusted dynamically depending on actual change in the policy in past iterations. PPO optimizes an alternative surrogate loss $J_{\text{PPO}}(\theta) = \mathbb{E}_{\tau_{\theta_{\text{old}}}}\left[\sum_t \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t - \lambda \cdot \text{KL}[\pi_{\text{old}}|\pi_\theta]\right]$, where λ is adjusted if the actual KL falls out of a target range. Our SURREAL-PPO implementation is a distributed variant of the PPO algorithm with the adaptive KL-penalty [78].

Evolution Strategies

Evolution Strategies (ES) [186] seeks to directly optimize the average sum of rewards. ES belongs to the family of Natural Evolution Strategies (NES) [237] and draws on intuition of natural evolution: a population of model parameters are maintained; at each iteration, each set of parameters are perturbed and evaluated; and models with the highest episodic return is recombined to be the base parameters of next iteration.

Specifically, we let θ be our model parameters with population distribution $p_\psi(\theta)$ parametrized by ψ. We can then take gradient step on ψ to optimize $J(\theta)$: $\nabla_\psi \mathbb{E}_{\theta\sim p_\psi} J(\theta) = \mathbb{E}_{\theta\sim p_\psi}\{J(\theta)\nabla_\psi \log p_\psi(\theta)\}$. Following recent work [186], we can reparametrize our population with mean parameter θ and

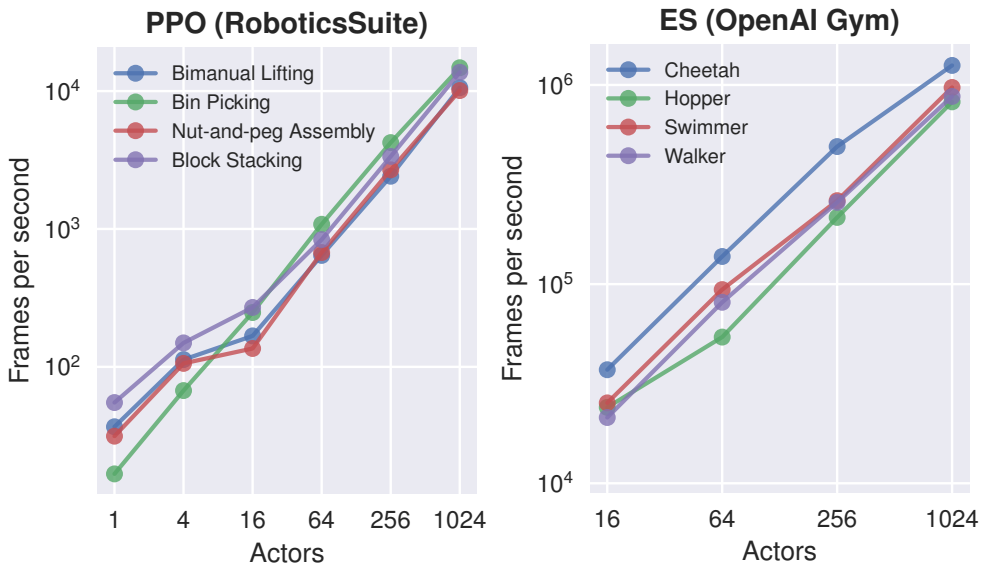


Figure 3.3: Scalability of environment interactions (in FPS) with respect to the number of actors on SURREAL-PPO (Robotics Suite) and SURREAL-ES (OpenAI Gym).

Gaussian noise $\epsilon \sim N(0, I)$ scaled by σ as perturbations to the mean parameters. It allows us to estimate gradient of the return as follows:

$$\nabla_\theta \mathbb{E}_{\epsilon \sim N(0,I)} J(\theta + \sigma\epsilon) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,I)} \{J(\theta + \sigma\epsilon)\epsilon\}$$

ES is intrinsically different from PPO. We highlight two major distinctions: First, as shown in the equation above, no gradient is computed for our step estimate. Hence, specialized hardware like GPUs has a less impact on ES than PPO. Second, as suggested by recent work [118], ES does not seek to optimize each and every actor's performance as in PPO. Instead, ES optimizes the average rewards of the entire population, thereby yielding robust policies that are less sensitive to perturbations.

3.4 Interface Design for Agile Research

The SURREAL-SYSTEM stack facilitates the development and deployment of distributed learning algorithms from the ground up. In this section, we provide a walk-through of our system implementation from an end-user's perspective. We will start with a bare-bone cloud account, walk up the ladders of abstractions, and reach the end goal of implementing distributed RL algorithms.

3.4.1 Provision a Computing Cluster

SURREAL reaches its full potential when running on a Kubernetes cluster. CLOUDWISE is a helper for setting up a Kubernetes cluster suitable for RL runtime. As shown in Fig. 3.8a, CLOUDWISE allows intuitive user interactions and generates a terraform definition which can be used to boot up the cluster. We also provide terraform files defining the cluster used in our experiments.

3.4.2 Defining a Distributed Experiment

SYMPHONY provides a simple way of declaring a distributed environment. Each distributed experiment is made up of multiple processes. They are declared in the launch script. Network communication patterns are specified by letting each process declare the services it provides and the services it binds to. An example can be seen in Fig. 3.8b.

Thanks to these platform-agnostic communication patterns, launch mechanisms in SYMPHONY can automatically configure different platforms to expose the necessary network interfaces. For example, SYMPHONY would create and configure platform-specific "services" when Kubernetes or Docker compose is used. It will assign local port numbers in Tmux mode. These arranged addresses are provided in the form of environment variables. See Fig. 3.8c for an example. This design allows smooth transition from local, small scale development to cloud-based large-scale deployments.

3.4.3 Scheduling Processes Flexibly

SYMPHONY provides bindings with various platforms to enable flexible scheduling (see Fig. 3.8d). Allowing processes to claim sufficient amounts of resources ensures that components run at full speed. Being able to control how much resource to allocate can drastically improve efficiency of algorithms. For instance, we demonstrate in Sec. 3.5.4 that running multiple actors on the same GPU can substantially improve the resource utility of a learning algorithm.

3.4.4 Dockerizing for Reproducibility

SURREAL experiments on Kubernetes are always dockerized to ensure scalability. This guarantees good reproducibility as the code for every launched experiment resides in a specific docker image in the registry. Moreover, SYMPHONY provides serialization for experiment declaration, saving not only source code but also the launching scripts. To ease the process of building docker images, SYMPHONY provides docker building utilities. For example, one can assemble files from multiple locations and build them into a single docker image.

3.4.5 Managing Experiments Conveniently

SYMPHONY also provides utilities to manage multiple experiments running on the same cluster. An example can be seen in Fig. 3.8e. One can view all running experiments, view all running processes of an experiment, and view logs of each process. These functionalities also allow team members to cooperate and exchange progress.

3.4.6 Running Surreal Algorithms

SURREAL algorithms are developed with tools provided by CLOUDWISE, CARAML, and SYMPHONY. SURREAL is designed to be easily extensible. One can implement a new `Learner` subclass with different model architectures and different learning schemes. The `Launcher` class provides a unified interface such that custom-built SURREAL components can be properly executed by scheduling code (see Fig. 3.8f).

3.5 Systems Benchmarking

Built to facilitate large distributed computation, SURREAL-SYSTEM allows algorithms to utilize large amounts of computing power. In this section, we investigate the system scalability with the SURREAL learning algorithms.

3.5.1 Evaluation Tasks

To examine the quality of our implementation of SURREAL-SYSTEM, we examine the performance and efficiency of SURREAL-PPO and SURREAL-ES in solving challenging continuous control tasks with these two algorithms.

We evaluate our SURREAL-PPO implementation on four robot manipulation tasks in Robotics Suite [208]. These tasks consist of tabletop manipulation tasks with single-arm and bimanual robots. We provide screenshots of these manipulation tasks in Fig. 3.5. All four tasks are complex and multi-stage, posing a significant challenge for exploration. In particular, we evaluate the strengths of SURREAL-PPO in learning visuomotor policies from pixel to control. The neural network policy takes as input RGB images and proprioceptive features and produces joint velocity commands at 10Hz. We evaluate our SURREAL-ES implementation on locomotion tasks in OpenAI Gym [15]. We report quantitative results on four locomotion tasks, including HalfCheetah, Hopper, Swimmer, and Walker2d. These standard tasks are used by prior work [186] for benchmarking Evolution Strategies implementations.

# Replay Shards	Throughput ($\times 10^3$ observations/s)
1	3.5
3	4.5
5	4.5

Table 3.2: Effect of sharding the replay buffer when using SURREAL-DDPG with 128 actors. Throughput measures the total number of actor observations received by the replay server.

3.5.2 System Scalability

We run our experiments on Google Cloud Kubernetes Engine. SURREAL-PPO actors for the Robotics Suite tasks were trained with 1 learner on a machine with 8 CPUs and a Nvidia V100 GPU, accompanied by 1, 4, 16, 64, 256, or 1024 actors. Multiple actors were run on the same machine in order to fully utilize the GPU (see Sec. 3.4 for details); for tasks Block Stacking, Nut-and-peg Assembly, and Bimanual Lifting, we place 16 actors on each machine of 8 CPUs and 1 Nvidia P100 GPU. For the Bin Picking task, we place 8 actors on each machine due to memory constraints. For OpenAI Gym were the actor nodes do not need a GPU for rendering and hence we use a dedicated 2-CPU machine per actor. The SURREAL-ES experiments were run on CPU only machines with 32 cores each with up to 32 actors batched per machine.

We measure SURREAL-PPO's scalability in terms of total actor throughput, which is the total environment frames generated by all actors combined per second. Fig. 3.3 shows our total actor throughput on the Block Stacking task. We see an approximately linear increase in total frames generated with an increase in actors.

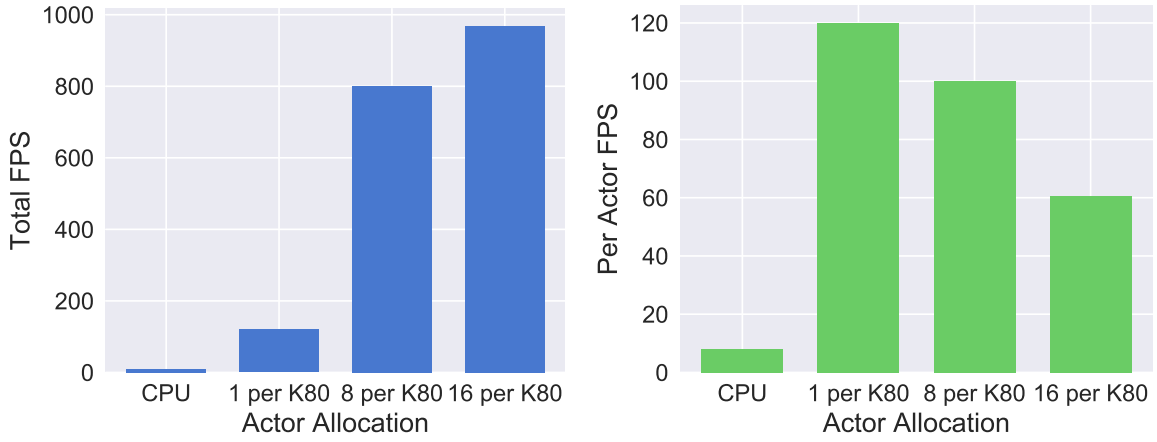
We measure SURREAL-ES's scalability in terms of total environments interactions per second. Fig. 3.3 shows an almost linear scaling of the same with the number of actors on various Gym environments.

3.5.3 Load Balanced Replay

As described in Sec. 3.3.2, the combination of SYMPHONY, CARAML, and Kubernetes enables simple and effective horizontal scalability. One example is sharding replay when training SURREAL-DDPG on the Gym Cheetah environment. Table 3.2 shows the number of experiences handled in total with 1, 3, and 5 sharded replays in presence of 128 actors. A single replay buffer can no longer handle all the actor outputs, becoming the bottleneck of the system. Three load-balanced replay buffers resolves congestion. More replays does not further improve overall throughput.

3.5.4 Batching Actors

As described in Sec. 3.4.3, SURREAL-SYSTEM supports a flexible scheduling scheme. It allows us to utilize resources more efficiently. We demonstrate this point by using SYMPHONY to batch multiple



(a) Total throughput

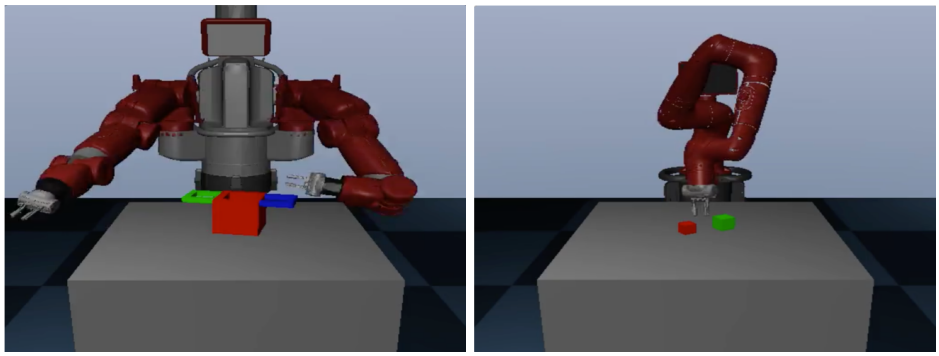
(b) Throughput per actor

Figure 3.4: Speed of policy evaluation on the Block Lifting task with pixel observations. We compare system speed with and without GPU acceleration. In the cases where a NVIDIA K80 GPU is used, we consider sharing the GPU among multiple actors. **a) Total throughput.** The number of environment frames collected by all actors. **b) Throughput per actor.** The number of environment frames collected by a single actor.

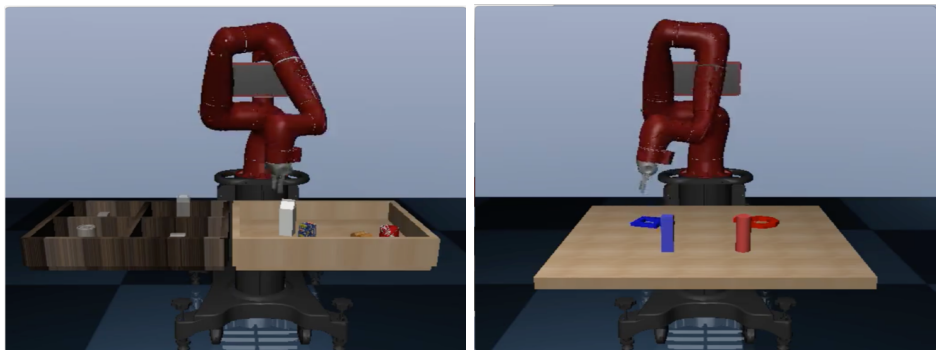
SURREAL-PPO actors on the same GPU. Fig. 3.4 shows policy evaluation speed on vision-based Block Lifting tasks, measured by environment frames per second. Each actor gets 1, $\frac{1}{8}$, or $\frac{1}{16}$ of a NVIDIA K80 GPU. Actor speed on CPU is provided for reference. Fig. 3.4a shows per GPU *throughput* measured by the total number of frames generated by all actors on the same GPU. Higher throughput means that experiments are more resource-efficient. Sharing a GPU among multiple actors achieves a better throughput and thus increases GPU utilization. However, as seen in Fig. 3.4b, each actor's speed decreases with an increasing number of actors sharing the same GPU. We set 16 actors per GPU in our experiments to attain the best trade-off between number of actors and per-actor throughput.

3.5.5 Serialization and Communication

In distributed RL algorithms, feeding large amounts of training data to the learner requires fast network communication and fast serialization. Doing everything on the main Python process would bottleneck the entire system. CARAML offloads communication to separate processes to circumvent global interpreter lock. It then serializes data and saves them to a shared memory to minimize interprocess data transfer. PyArrow is used to serialize data because it provides fast deserialization. The speed-up of these optimizations is measured on reinforcement learning algorithms SURREAL-DDPG and SURREAL-PPO trained on the Gym Cheetah environment. Speed-up measured by learner iterations per second is reported in Table. 3.3. The speed-up in DDPG is more pronounced



(a) Bimanual Lifting (b) Block Stacking



(c) Bin Picking (d) Nut-and-peg Assembly

Figure 3.5: Screenshots of our robot manipulation tasks in Robotics Suite [208]: **a) Bimanual Lifting.** The goal is to grab the pot by both handles and lift it off the table; **b) Block Stacking.** The robot picks up the red block and places it on top of the green one; **c) Bin Picking.** The robot picks up each item and place each into its corresponding bin; **d) Nut-and-peg Assembly.** The goal is to place the nut over and around the corresponding pegs.

as PPO is less communication bound than DDPG. In comparison, the PPO learner iteration performs more computation than that of DDPG.

3.6 Quantitative Evaluation

Here we further examine the effectiveness and scalability of SURREAL-SYSTEM in learning efficiency and agent performance. To this end, we implement the Proximal Policy Optimization (PPO) and Evolution Strategies (ES) algorithms using the APIs provided by our framework. These two algorithms have distinct characteristics: ES is embarrassingly parallel with a large number of homogenous actors, while PPO requires the coordination between heterogeneous types of parallel components. Our primary goal is to answer the following two questions: 1) are our implementations of these two algorithms capable of solving challenging continuous control tasks and achieve higher performances

	PPO (iters/s)	DDPG (iters/s)
without optimization	5.1	5.4
with optimization	**38**	**54**

Table 3.3: Learner iteration speed for PPO and DDPG with and without the communication optimizations of CARAML. Numbers are obtained when training on the Gym Cheetah environment.

than prior implementations, and 2) how well can our distributed algorithms scale up with an increasing amount of computational resources? The experiment setup used here is same as the one described in Sec. 2.4.

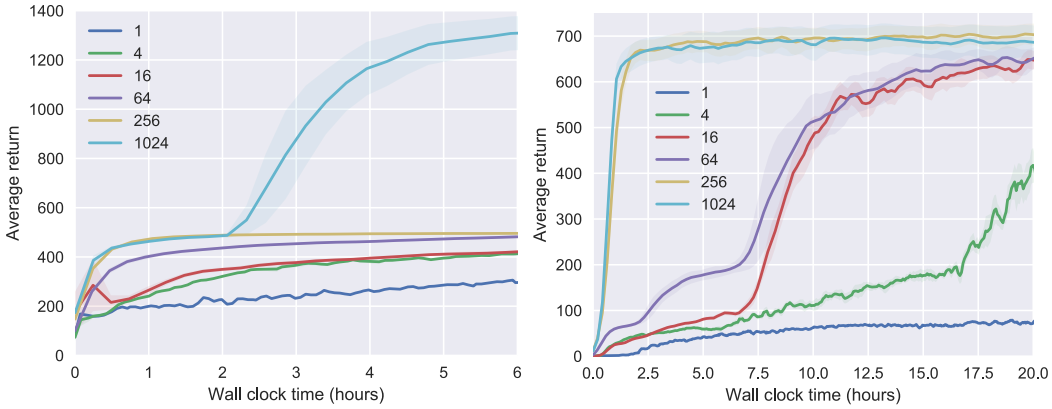
3.6.1 Surreal-PPO Evaluation

For each of the Robotics Suite tasks, we train a PPO model which takes as input an $83 \times 83 \times 3$ RGB image and proprioceptive features (e.g., arm joint positions and velocities). The image is passed through a convolutional encoder. The resulting activations are flattened to a vector and concatenated to the proprioceptive features, which is further fed through an LSTM layer of 100 hidden units. The output of the LSTM layer is passed through additional fully-connected layers of the actor network and the critic network for producing the final outputs.

Fig. 3.6 reports results of our PPO implementations with varying number of actors. We see an overall trend towards higher scores and faster training with an increased number of actors. In the Bimanual Lifting and Bin Picking tasks, we observe a substantial benefit of using 1024 actors, where the learned policy is able to advance to later stages of the tasks than the runs with fewer actors. The trained PPO model is able to pick up the can and place it into the bin in the Bin Picking task, and in the Bimanual Lifting task is able to lift the pot off the table. In the Block Stacking and Nut-and-peg Assembly tasks, the experiments with 256 actors and 1024 actors managed to converge to the same level of performance, which achieve both faster and better learning than the ones with fewer actors.

3.6.2 Surreal-ES Evaluation

Our ES implementation takes low-dimension state features as input. These features are passed through a network with two fully-connected layers. The activations are passed through a final fully-connected layer and tanh activation to map to the predefined action space. To interact with the environment, we use a stochastic policy with mean generated from the neural network and fixed standard deviation of 0.01. For the population noise, we sample from Gaussian distribution centered at 0 with standard deviation 0.02. Additionally, we discretize the action space to 10 bins per component for Swimmer and Hopper to aid exploration.



(a) Bimanual Lifting

(b) Block Stacking

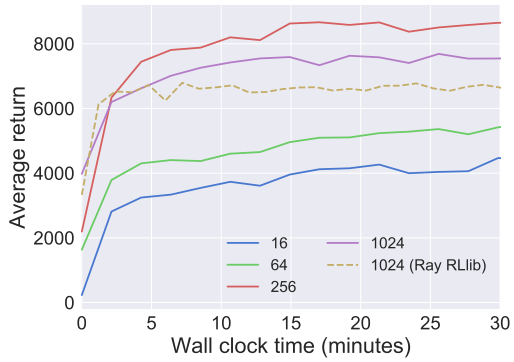
(c) Bin Picking

(d) Nut-and-peg Assembly

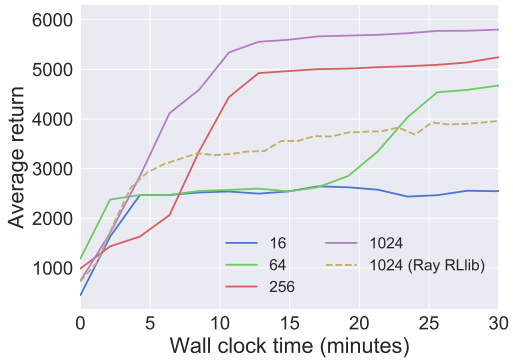
Figure 3.6: Scalability for SURREAL-PPO experiments on Robotics Suite tasks. The solid line represents mean return and the translucent region around the mean represents one standard deviation. PPO models were trained on 1, 4, 16, 64, 256, and 1024 actors.

Fig. 3.6 shows performances of our ES implementation on various locomotion tasks with increasing number of actors. We see that SURREAL-ES scales well with a large number of actors, leading to faster convergence and shorter training time. We hypothesize that it is attributed to the effects of improved exploration with more actors executing diverse policies. Through experimentation, we find that the scale of standard deviation of Gaussian noise compared to the scale of parameters greatly impacts learning efficacy. High noise values cause learning to be unstable whereas low noise values cause slow learning.

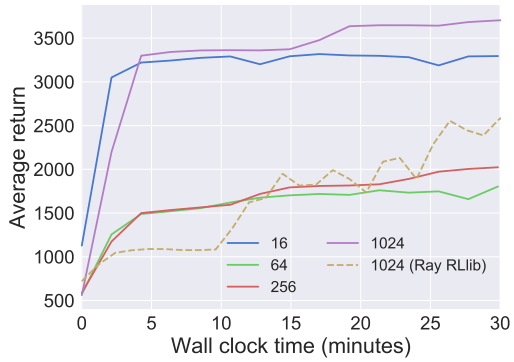
SURREAL-ES is very competitive with the state-of-the-art reference implementation [124] as shown in Fig 3.7. Across all four Gym environments, SURREAL-ES outperforms Ray RLlib in terms of both final performance and wall-clock time.



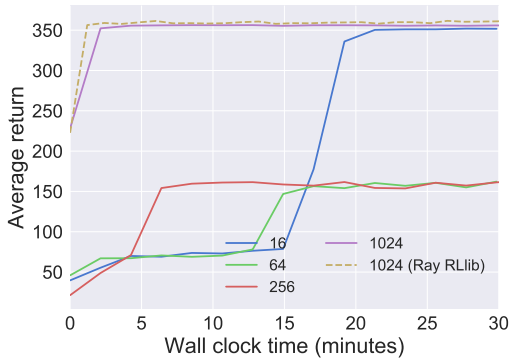
(a) HalfCheetah



(b) Walker2d



(c) Hopper



(d) Swimmer

Figure 3.7: Scalability of SURREAL-ES on OpenAI Gym tasks. The dashed line represents the best curve obtained by Ray RLlib [124] with 1024 actors and the rest correspond to our implementation trained with 16, 64, 256, 1024 actors.

3.7 Conclusion

We introduced SURREAL-SYSTEM, a four-layer distributed learning stack for reproducible, flexible, scalable reinforcement learning. It enables an end-user to deploy large-scale experiments with thousands of CPUs and hundreds of GPUs, facilitating researchers to rapidly develop new distributed RL algorithms while reducing the effort for configuring and managing the underlying computing platforms. We have released the complete source code of our SURREAL framework (`https://github.com/SurrealAI`) to the research community. We hope that this project could facilitate reproducible research in distributed reinforcement learning.

```
>> python cloudwise-gke.py
Please give your cluster a name
> kuflexes
Do you wish to create a node pool
    with 32 CPUs and 1 Nvidia V100
     GPU per machine?
> Yes
...
Generating kuflexes.tf.json
Use "terraform apply" to create
    the cluster
```

(a) Interacting with CLOUDWISE CLI to generate cluster specification.

```
# launch_experiment.py
exp = symphony.Experiment()
learner = exp.new_process(cmd="
    python learner.py")
replay = exp.new_process(cmd="
    python replay.py")
replay.binds("replay")
learner.connects("replay")
cluster.launch(exp)
...
```

(b) Declaring a distributed experiment using SYMPHONY. Network connnections are defined in a straightforward way.

```
# learner.py
host = os.environ["
    SYMPH_REPLAY_HOST"]
port = os.environ["
    SYMPH_REPLAY_PORT"]
sender = caraml.Sender(host=host,
    port=port, serializer=pickle.
    dumps)
sender.send(data)
# replay.py
...
receiver = caraml.Receiver(host=
    host, port=port, serializer=
    pickle.loads)
data = receiver.recv()
```

(c) SYMPHONY maps network address to environment variables so network connection is compatible across platforms.

```
# launch_experiment.py
exp = symphony.Experiment()
learner = exp.new_process(cmd="
    python learner.py")
dispatcher = symphony.Dispathcer(
    cluster= "kuflexes.tf.json")
dispatcher.assign_to_nodepool(
    learner,
     "v100-nodepool", cpu=5, gpu
         =1)
```

(d) SYMPHONY provides scheduling bindings to underlying clusters.

```
# monitoring experiments
>> kuflexes list-experiments
Humanoid
Cheetah
>> kuflexes switch-experiment
    Cheetah
>> kuflexes list-processes
actor-0
...
>> kuflexes logs actor-0
actor-0 running ...
```

(e) SYMPHONY provides experiment management features to view experiments and processes.

```
# ppo_better.py
class BetterPPOLearner(PPOLearner)
    :
        ...

# launch_ppo_better.py
from ppo_better import
    BetterPPOLearner
launcher = surreal.PPOLauncher(
    learner=BetterPPOLearner)
launcher.main()
```

(f) SURREAL is designed to be easily extensible. Each component can be subclassed to show custom behaviors.

Figure 3.8: Code snippets for various SURREAL use cases. The SURREAL-SYSTEM stack provides support for cluster creation, experiment declaration, network communication, resource allocation, experiment management and extensions.

Chapter 4

Realistic Simulation for Embodied Visual Agents

4.1 Introduction

Simulation environments have proliferated over the last few years as a way to train robots and interactive agents in a rapid and safe manner. In these environments, agents learn to engage in physical interactions [126, 120], navigate based on sensor signals [138, 157, 264, 196, 27], or plan long-horizon tasks [61, 247, 240, 122]. In simulation, agents learn to perform interactions that actively change the input sensor signals and the state of the environment towards a desired configuration, capabilities at the core of what an embodied agent needs to achieve.

However, existing simulation environments that combine physics simulation and robotic tasks often cater to a narrow set of tasks and include only clean, small-scale scenes [93, 253, 224, 244, 117, 266]. The few simulation environments that include large scenes such as homes or offices either disable the possibility of interacting with the scene, focusing only on navigation (e.g. Habitat [134]), or use simplified modes of interaction (e.g. AI2Thor [105], VirtualHome [168]). These simulators do not support the development of end-to-end sensorimotor control loops for tasks that require rich, continuous interaction with the scene. Such tasks are difficult to accomplish in the aforementioned simulators, and the simplified modes of interaction lead to difficulties in transferring the learned policy onto real robots.

We present *iGibson 1.0* (alternative called just *iGibson* in this manuscript), a novel simulation environment that enables the development of embodied agents for interactive tasks in large-scale realistic scenes (Fig. 4.1). Interactivity is achieved by leveraging a physics engine processing all elements in the scene, enabling manipulation of rigid and articulated objects as well as mobility. iGibson 1.0 aims at unifying several aspects of robot simulation that are often available in different

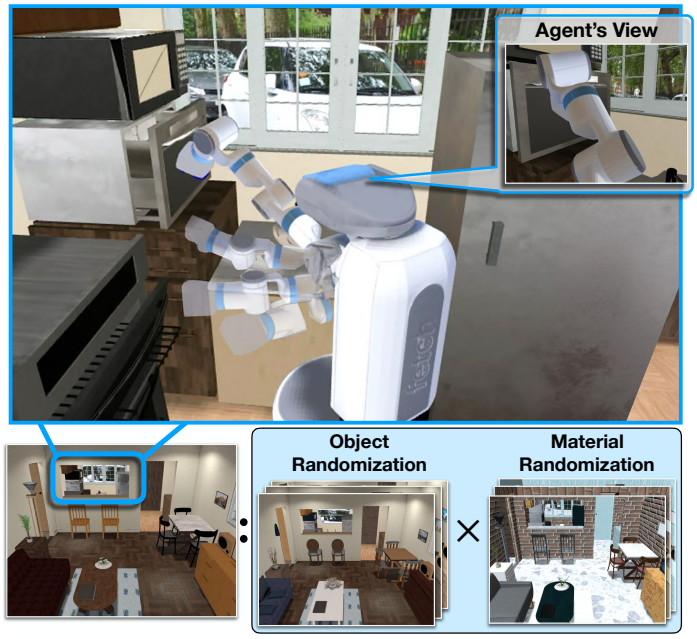


Figure 4.1: Robot performs an interactive task in iGibson 1.0. It operates in the kitchen of one of iGibson's fully interactive scenes, planning an interaction with the arm using a integrated sampling-based motion planner and receiving first-person view. *Bottom:* The same scene can be randomized with different materials and/or object models

software tools, such as physics simulation for interaction with objects and robot control, high-quality simulated sensors, integration with reinforcement learning frameworks, and realistic indoor scenes that reflect the objects distribution of real homes. This integration allows fully physics based simulation of robot tasks (i.e. simulating the full complexity of the task) and allows developing task and motion planning, reinforcement learning or imitation learning solutions for those tasks with virtual sensor signals.

iGibson 1.0 contains 15 fully interactive and visually-realistic scenes with a total of 108 rooms. These scenes were generated by annotating 3D reconstructions of real-world scans (static scenes represented by a single mesh) and converting them into fully interactive scene models (scenes filled with articulated object models). In this process, we respect the original object-instance layout and object-category distribution. The object models are extended from open-source datasets [24, 233, 244] enriched with annotations of material and dynamic properties. iGibson's physics-based renderer leverages the extra information provided in the material annotation (maps of metallic, roughness and normals) to generate high-quality virtual images. To further facilitate the training of more robust visuomotor agents, iGibson 1.0 offers domain randomization procedures for materials (both visual appearances and dynamics properties) and object shapes while respecting the distribution of object placements and preserving interactability. iGibson 1.0 is also equipped with a graphical user

interface that allows human users to easily interact with the scenes, enabling efficient collection of human demonstrations for imitation learning.

In summary, iGibson 1.0 provides the following novel features that facilitate developing and training robotic solutions:

1. Fifteen fully interactive scenes containing 108 rooms modelled after real world homes with articulated object models annotated with materials and dynamics properties. Additionally, we support importing CubiCasa5K [100] and 3D-Front [57] layouts, giving access to more than 12000 additional fully interactive scenes.

2. Realistic virtual sensor signals, including a physics-based renderer (PBR) for RGB images, rendering of normals, depth, point clouds, virtual LiDAR signals, and optical/scene flow. We further integrate domain randomization functionality (visual texture, dynamics properties and object instances) that facilitates generalization to unseen scenes.

3. Useful tooling for developing robotics solutions in simulation, such as a human-computer interface for humans to provide interactive demonstrations, and sampling-based motion planners for navigation and manipulation.

We demonstrate the benefits of these novel features in a comprehensive set of experiments in which visual agents are trained for navigation and interactive tasks. Our experiments show that iGibson 1.0 enables researchers to 1) train more robust and generalizable sensorimotor policies thanks to its realistic virtual sensor signals (including LiDAR) and domain randomization mechanisms, 2) collect human demonstrations and train imitation learning policies for manipulation and mobile manipulation tasks, and 3) learn intermediate visual representations linked to interactability of the scene that accelerate training of downstream manipulation tasks. iGibson 1.0 is open-source and academically developed, and available on our website http://svl.stanford.edu/igibson/.

4.2 Related Work

The use of simulation in robotics has significantly increased in recent years and the research community has proposed a number of simulators for robotics and embodied AI. Here, we use the terms physics simulator and simulation environment as follows. A **physics simulator** is an engine capable of computing the physical effect of actions on an environment (e.g. motion of bodies when a force is applied, or flow of liquid particles when being poured) [38, 220, 114, 215, 125, 153]. On the other hand, a **simulation environment** is a framework that includes a physics simulator, a renderer of virtual signals, and a set of assets (i.e. models of scenes, objects, robots) ready to be used to study and develop solutions for different tasks. Both components are crucial for advancing embodied AI and robotics. Here, we focus on the discussion of simulation environments.

	iGibson 1.0 (ours)	Gibson [25]	Habitat [18]	Sapien [15]	AI2Thor [19]	VirtualH [20]	TDW [26]
Provided Large Scenes Real-World / Designed	15 homes (108 rooms) / –	1400 / –	–	–	– / 120 rooms	– / 7	– / 25
Provided Objects Number / Materials	570 / Yes	– / –	– / –	2346 / No	609 / Yes	308 / No	200 / Yes
Agent/World Interaction **F**orces, **P**redefined **A**ctions	F	–	–	F	F & PA	F & PA	F
Physics Engine	Bullet	Bullet	Bullet	PhysX	Unity	Unity	Unity & Flex
Type of Simulation **R**igid **B**odies **P**hysics, **R**igid **B**odies with **PA**, **P**articles and **F**luids	RBP	RBP	RBP	RBP	RBP&RBPA	RBPA	RBP&PF
Supported Task	Nav.&Manip.	Nav.	Nav.	Nav.&Manip.	Nav.&Manip.	Nav.&Manip.	Nav.&Manip.
Type of Rendering	PBR	IBR	PBR	PBR,RTX	PBR	PBR	PBR
Virtual Sensor Signals	RGB,D,N SS,FL,LiDAR	RGB,D N,SS	RGB,D,SS,S	RGB,D,SS	RGB,D,SS,S	RGB,D SS,FL	RGB,D,SS,S
Domain Randomization **S**cene,**O**bject,**M**aterials	S,O,M	-	-	-	S	S	S,O
Speed	++	+	+++	++(PBR)/-(RTX)	+	+	+
Human Interface	Mouse Keyboard	-	Mouse Keyboard	-	Mouse Keyboard	Natural Language	Virtual Reality
Integrated Motion Planner	Yes	No	No	No	No	No	No
Specialty	Phys. Int. in Large Scenes	Nav.	Fast, Nav.	Articulation, Ray Tracing	Object States, Task Planning	Object States, Task Planning	Audio, Fluids

Type of rendering: PBR:Physics-Based Rendering, **IBR**:Image-Based Rendering, **RTX**:Ray Tracing
Virtual sensor signals: RGB: Color Images, **D**:Depth, **N**:Normals, **SS**:Semantic Segmentation, **LiDAR**:Lidar, **FL**:Flow (optical and/or scene), **S**: Sounds

Figure 4.2: Comparison of Simulation Environments

Several simulation environments have been proposed recently to study manipulation with stationary arms [253, 93, 266, 48, 117]. Most of them are based on Bullet [38] or MuJoCo [220] for physics simulation, and render images with the default renderer or a Unity [215] plugin. Different from these simulation environments, iGibson focuses on large-scale (house-size) scenes and includes fifteen fully interactive scenes with 108 rooms – such as kitchens and bedrooms – where researchers can develop solutions for navigation, manipulation and mobile manipulation.

Closer to iGibson are simulation environments that include large-scale realistic scenes (e.g. homes or offices), which we summarize and compare in Fig 4.2. Gibson [243] (now Gibson v1) was the precursor of iGibson. It includes over 1400 3D-reconstructed floors of homes and offices with real-world object distribution and layout. Although Gibson incorporates PyBullet as its physics engine for simulating robot navigation, each scene is one single fully rigid object (static mesh). Thus, it does not allow agents to interact with the scenes other than collisions with the mesh, restricting its use to only navigation.

A similar environment is Habitat [134]. Despite its high rendering speed, Habitat uses the non-interactive assets from Gibson v1 [243] and Matterport [23] and therefore only supports navigation tasks and simplified rearrangement of added objects. Recent work [242] introduced an extension to the Gibson v1 environment to support Interactive Navigation [242] where parts of the reconstructions corresponding to five object classes (chairs, tables, desks, sofas, and doors) in several Gibson static models were segmented and replaced with interactive versions. This enabled navigation agents to interact with the scene, and thus allowed for the first benchmark for Interactive Navigation. However, the simulators above are not interactive [243, 134] or partially interactive [242]. iGibson

encapsulates the functionalities in all previous versions of Gibson with backwards compatibility for the static environments.

A variety of simulation environments have been proposed recently for scene-level interactive tasks, such as Sapien [244], AI2Thor [105], VirtualHome [168], and ThreeDWorld (TDW [59]). These simulators adopt different ways of agent-world interactions. **Predefined Actions (PA)** consist in the set of actions that can be performed for each object type. When the agent is close enough to an object and the object is in the right state (precondition), the agent can select a predefined action, and the object is "transitioned" to the next state (postcondition). In Fig 4.2 we refer to this technique as Rigid Bodies with Predefined Actions (RBPA). It is possible to combine Rigid Bodies with Predefined Actions (RBPA) and Rigid Body Physics (RBP), such as first using RBPA to grasp an object and then using RBP after releasing it. AI2Thor and VirtualHome use predefined actions (PA) as an abstraction of physical interactions and allow agents to modify the object states instantaneously (e.g. open a closed cabinet), suitable to study high-level task planning.

While the availability of PA helps focusing on symbolic reasoning, the full robotics problems require RBP to simulate the tasks in all its complexity. PA limits access to all granularities of the task, which impedes robot learning and sim2real transfer. With the purpose of simulating full robotics tasks, iGibson uses RBP, simulating the physics behavior of all objects continuously, with embodiment of real robots. This is crucial if the learned policy is to be deployed in the real world. Similar to iGibson, Sapiens also uses RBP without PA, but with smaller scenes, focusing on interaction with articulated objects. In iGibson, we enrich the articulated objects models from the PartNet-Mobility dataset introduced by Sapien with materials and dynamics properties. TDW is also capable of continuous RBP, with simplifications that facilitate grasping using robot avatars and not real robot platforms.

While other simulation environments focus on particular aspects of embodied agent simulation (Fig. 4.2), iGibson **uniquely** unifies a set of important tools for robotics that together enable robot learning in large scenes: support of LiDAR and PBR rendering, speed that enable reinforcement learning, robot integration (URDF, controllers, motion planners), and continuous physics simulation of agents and objects. This integration of features enables tasks such as mobile manipulation, illustrated in Fig. 4.3, including physics-based robot simulation across the entire task.

4.3 iGibson Simulation Environment

In this section, we discuss the main structure, properties and features of iGibson that support training of robust sensor-guided policies for navigation and manipulation.

(a) pick (b) transport (c) place

Figure 4.3: Simulated Fetch Robot performing a mobile manipulation task in a cluttered environment.

4.3.1 Simulation Characteristics and API

At the highest level, iGibson follows the OpenAI Gym [15] convention. The environment receives an action and returns a new observation, reward and additional meta-information (e.g. if the episode has ended). Environments are specified with config files that determine scenes, tasks, robot embodiments, sensors, etc. Given a config file, iGibson creates an `Environment` that contains a `Task` and a `Simulator`. The `Simulator` contains a `Scene`, with a list of interactive `Objects` and one or more `Robot` instances. It also contains a `Renderer` that generates virtual visual signals from any point of view, such as a camera mounted on a robot or an external third person view. The `Task` defines the reward, initial and final conditions for the scene and the agents. While modular and easy to extend, most users may only need to interface with `Environment` after instantiating it with the appropriate config files.

iGibson comes with multiple easy-to-use configs, demos and Docker [137] files. It has been extensively adopted to train visuo-motor policies that successfully transfer to the real world [136, 135, 101, 83], and was the platform for iGibson Sim2Real Challenge at CVPR20 [155] and iGibson Challenge at CVPR21. The provided virtual LiDAR sensor has been used for robotics research in planning and reinforcement learning for social navigation and mobile manipulation [240]. iGibson is easily parallelizable and supports off-screen rendering on clusters.

4.3.2 Fully Interactive Assets

iGibson provides fifteen high quality fully interactive scenes with 108 rooms (see Fig. 4.4), populated with interactable objects. The scenes are interactive versions of fifteen 3D reconstructed scenes included in the Gibson v1 dataset. To preserve the real-world layout and distribution of objects, we follow a semi-automatic annotation procedure. This process is radically different from the annotation we performed for the Interactive Gibson Benchmark [242]. Instead of segmenting the original scene and replacing part of the meshes with interactive object models, we create fully interactive

Figure 4.4: Fifteen interactive iGibson 1.0 scenes modelled after real-world reconstructions, preserving layout, distribution and size of objects.

counterparts of the 3D reconstruction from scratch. This eliminates the need to fix artifacts in the original mesh due to reconstruction noise or segmentation error, and allows us to improve the overall quality of the scenes.

The scene generation process is composed of two annotation phases. First, the layout of the scene is annotated with floors, walls, doors and window openings. Then, all objects are annotated with 3D bounding boxes and class labels. We annotate bounding boxes for 57 different object classes, including all furniture types (doors, chairs, tables, cabinets, TVs, shelves, stoves, sinks, etc) and some small objects (plants, laptops, speakers, etc); see project website for the complete list. Annotating class-labeled bounding boxes allows us to scale and use different models of the same object class, while maintaining the real-world distribution of objects in the scene. In this way, we are able generate realistic randomized versions of the scenes (see Sec. 4.3.4). To achieve the highest quality, for each class-labeled bounding box, we select a best fitting object model. The scene is also annotated with lights, with which we generate light probes for physics-based rendering (see Sec. 4.3.3). We also bake in a realistic ray-traced ambient light and other light effects in the walls, floors and ceilings.

The object models are curated from open-source datasets: ShapeNet [24], PartNet Mobility [244, 143], and SketchFab. To preserve visual realism of the original reconstruction, we improve the object visual quality by annotating different parts of the models with photorealistic materials, which are then used by iGibson's physics-based renderer. We utilize materials from CC0Texture, including wood, marble, metal, etc. To achieve a high degree of physics realism, we curate a mapping from visual materials to friction coefficients. We additionally compute the collision mesh, center of mass and inertia frame for each link of all objects. To assign realistic mass and density for different objects, we take the the median values of the top 20 search results from Amazon.

Additionally, we provide compatibility with CubiCasa5K [100] and 3D-Front [57] repositories of home scenes. We use their scene layouts and populate them with our annotated object models,

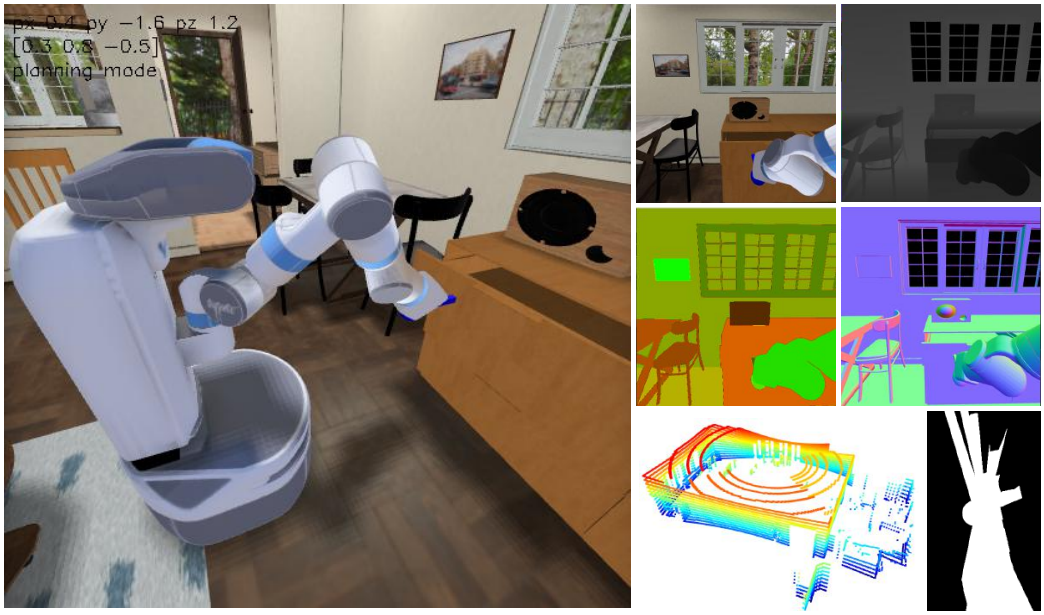


Figure 4.5: Robot interacting in iGibson 1.0 (large picture: 3rd person view) and **virtual sensor signals** generated. Policies and solutions can make use of the following channels: (from top to bottom, left to right) RGB, depth, semantic/instance segmentation, normals, 16D LiDAR (point cloud), 1D LiDAR (also as occupancy map). Not depicted: optical/scene flow, joint encoders for robot's and objects' joints, poses, wrenches, contact points, and map localization.

leading to additional more than 12000 interactive home scenes. These scenes contain fewer objects than the fifteen iGibson scenes, but provide a very large number of additional models to train tasks.

The fully interactive scenes we include in iGibson enable learning of interactive tasks in large realistic home scenes; in Sec. 4.4.3 we show that the scenes can be used to learn a useful visual representation that accelerates the learning of downstream manipulation tasks.

4.3.3 Virtual Sensors

A crucial component of iGibson is the generation of high quality virtual sensor signals, i.e. images and point clouds, for the simulated robots. In the following, we summarize the most relevant of these signal generators (Fig. 4.5).

Physics Based Rendering

In iGibson, we include an open-source physics-based renderer, which implements an approximation of BRDF models [191] with spatially varying material maps including roughness, metallic and tangent-space surface normals, extending [43].

LiDAR Sensing

Many real-world robots are equipped with LiDAR sensors for obstacle detection. In iGibson, we support virtual LiDAR signals, with both 1 beam (e.g. Hokuyo) and 16 beams (e.g. Velodyne VLP-16). We include a simple drop-out sensor noise model to emulate the common failure case in real sensors in which some of the laser pulses do not return. Additionally, we provide the functionality to turn the 1D LiDAR scans into local occupancy maps, which are bird's-eye view images with three types of pixels indicating free, occupied, or unknown space.

Additional Visual Channels

In addition to RGB and LiDAR, we support a wide range of visual modalities, such as depth maps, optical/scene flow and normals, segmentation of semantic class, instance, material and movable parts. These modalities can support research topics such as: depth/segmentation/normal/affordance prediction [167, 22, 245], action-conditioned flow prediction [149], multi-modal pose estimation [32, 228, 87], and visuomotor policy training assuming perfect vision systems [242, 249].

4.3.4 Domain Randomization

It is standard practice for robot learning to partially randomize the environment's parameters in order to make the policy more robust [185, 218, 94, 4]. With the model being trained in a wide distribution of environments, it will be more likely to generalize to unknown evaluation environments. The evaluation environment may be the real world if we aim to train in simulation and transfer the policy to a real robot. In iGibson, we include domain randomization that leads to an endless variation of visual appearance, dynamics properties and object instances with the same scene layout.

First, we provide **object randomization**. The original 3D reconstructions are annotated with class-labeled object bounding boxes. These labels can be used to instantiate any object model of the corresponding class into the given bounding box (e.g. a bounding box labeled as "table" can be filled with any table model). This randomization maintains the semantic layout of the scenes (i.e. the object categories remain at the same 3D locations) while enabling near-infinite combinations of object instances. It provides strong variation in depth maps and LiDAR signals that helps robustify policies based on these observations (see Sec. 4.4.1).

Second, we provide **material randomization**. In addition to high-quality material annotation for object and scene models, we provide a mechanism to randomize the specific material model associated with each object part (e.g. associating a different type of wood or metal). The effect is a stark color randomization that still represents plausible material combinations. This randomization generates strong variations in the RGB images and helps robustify policies based on this observation (see Sec. 4.4.1). Moreover, the dynamics properties of all object links can be randomized based on a curated mapping from visual materials to dynamics properties.

4.3.5 Motion Planning

Motion planners provide collision-free trajectories to move a robot from an initial to a final configuration [111]. They can be used to generate collision-free navigation paths for robot bases and collision-free motion paths for robot arms. In iGibson, we include implementations of the most popular sampling-based motion planners: rapidly growing random trees (RRT [110]) and its bidirectional variant (BiRRT [170]), and lazy probabilistic road-maps (lazyPRM [14]), adapted from [18]. Sampling-based motion planners can have rather suboptimal and intricate paths. To alleviate this, we include acceleration-bounded shortcuts [74] for smoother paths.

4.3.6 Human-iGibson Interface

We provide a human-iGibson interface that enables users to navigate and interact in iGibson scenes using mouse and key commands on a viewer window. The user can navigate and interact with (pull, push, pick and place) objects. While a virtual reality or a 3D mouse interface may provide a more intuitive experience, most users do not have the necessary hardware. This interface offers a natural and simple way to demonstrations for imitation learning, evaluate the difficulty or feasibility of a task, or change the scene into a better initial state, for example. This interface is also integrated with the motion planner to command the robot to desired base and/or arm configurations. We verify this interface facilitates efficient development of interactive robotic solutions in Sec. 4.4.2.

4.4 Experiments

The goal of our experiments is to study how iGibson's features help to develop AI agents. Specifically, we examine:

- (Sec.4.4.1) does iGibson's **domain randomization** and realistic virtual sensor signals (including **LiDAR**) allow navigation agents to generalize to unseen scenes (including the real world)?
- (Sec.4.4.2) can the **human-iGibson interface** be used to efficiently train imitation learning agents for manipulation and **mobile manipulation tasks**?
- (Sec.4.4.3) does the **full interactivity** in the scenes allow agents to learn visual representations that accelerate learning of downstream manipulation tasks?

4.4.1 Domain Randomization and Realistic Sensor Signals for Navigation

In the first three experiments, we evaluate the generalization benefits brought by our realistic virtual sensor signals (including LiDAR) and domain randomization. First, we compare the generalization

Figure 4.6: Robot navigating the real-world counterpart of the iGibson 1.0 scene `Rs_int`. The robot executes a policy trained in simulation with virtual LiDAR signals, without domain adaptation. The quality and realism of iGibson facilitates zero-shot policy transfer.

capabilities of vision-based reinforcement learning policies trained with and without domain randomization. Concretely, we evaluate the performance of policies trained in iGibson for PointGoal tasks [3] in held-out scenes with held-out visual textures. The observations for the policy include the depth maps, the robot's linear and angular velocities, the goal location in the robot's reference frame, and the next 10 waypoints in the shortest path between the robot's current location and the goal location, separated by 0.2 m. The waypoints are computed only based on the room layout, not the objects, so the robot mainly relies on depth maps for obstacle avoidance.

Second, we evaluate the performance of robot policies trained in iGibson to navigate to a target object (a lamp) using virtual RGB images also in held-out scenes with held-out visual textures. The task goal is to get at least 5% of the image occupied by the pixels of the target object. The observation for the policy only includes RGB images. In the above two experiments, we train in eleven scenes and evaluate in four held-out scenes with held-out visual textures.

Third, we evaluate the performance of the policies trained in iGibson for a PointGoal tasks [3] using **virtual LiDAR signals** with no vision inputs, and examine how well those policies transfer to the real world without adaptation, a hard test for generalization. The observations for the policy include a 1D LiDAR scan with 512 laser rays, the robot's linear and angular velocities, and the goal location in the robot's reference frame. To focus on sim2real transferability, we train in our scene `Rs_int`, for which we have access to the real-world counterpart (see Fig. 4.6).

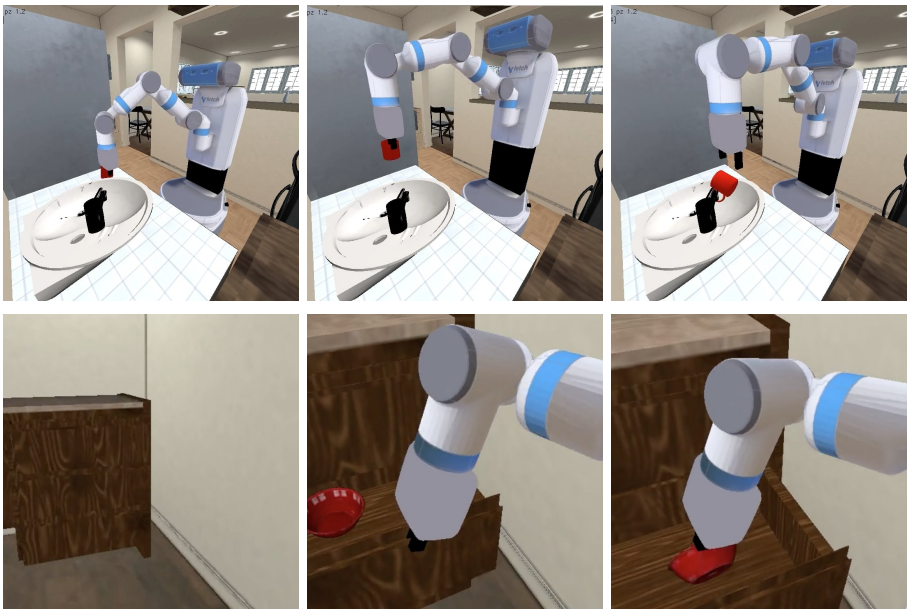


Figure 4.7: Imitation learning from human demonstration. *Top:* third-person view of a robot performing a pick-and-place task. The policy trained using demonstrations collected with our human-iGibson interface achieves 98% success rate. *Bottom:* first-person view of a robot performing a mobile manipulation task. The policy trained using teleoperated demonstrations achieved 70% success rate.

Results

In the first two experiments, we observe better generalization capabilities in policies using iGibson's domain randomization. For PointGoal navigation based on depth images, the performance goes from 0.27 to 0.40 SPL [3] and from 31.25% to 44.75% success rate when using randomization, indicating that the larger variety of shapes observed in the training process generates more robust depth-based policies. For object navigation based on RGB images, the performance goes from 49.75% to 57.5% success rate, indicating that material randomization helps in obtaining RGB-based policies that are more generalizable to unseen scenes and textures. Finally, for PointGoal navigation based on LiDAR signals, the policy achieves 33% success rate in `Rs_int` in iGibson, and 24% success rate in the real-world apartment. With only a 9% drop in performance and the failures mostly occurring in the same episodes (same pairs of the initial and goal locations in iGibson and real world), this experiment indicates that the LiDAR signals generated in iGibson are realistic enough to facilitate zero-shot policy transfer. In summary, as shown in Fig. 4.2, iGibson provides unique support to train with realistic virtual sensor signals (e.g. LiDAR) and domain randomization, which leads to more robust robot navigation policies that successfully transfer to novel scenes.

4.4.2 Imitation Learning of Manipulation

In the second set of experiments, we evaluate iGibson as platform to train robots to perform manipulation and mobile manipulation tasks with Imitation Learning. First, we test the usability of the human-iGibson interface to efficiently collect demonstrations of **manipulation-only** tasks. We collect 50 demonstrations of pick-and-place operations with 20 mug models: pick a mug and place in the sink (Fig. 4.7), and store pairs of state (object position) and action (desired end-effector translation). We use the demonstrations to train a behavioral cloning policy that maps states to actions at 20 Hz. The action space consists of two parts: a 3-dimensional continuous space for desired end-effector translation, and a 1-dimensional discrete space for gripper opening. The desired end-effector translation is computed using inverse kinematics (IK) and executed with a joint position controller. The evaluation is conducted using a simulated mobile manipulator (Fetch robot), and generalization is tested with 5 unseen mugs.

Second, we collect demonstrations for imitation learning through continuous teleoperation using a system similar to Roboturk [133] for **mobile-manipulation** tasks. We collect 200 demonstrations with a simulated Fetch robot for search-and-pick operations: the robot must navigate and interact with a cabinet to open drawers, find a bowl and pick it (Fig. 4.7 (bottom)). The bowl and robot initial poses are randomized between episodes. Using these demonstrations, we train a behavioral cloning policy mapping observations to actions at 20 Hz. The observation space includes the robot proprioceptive information (joint positions and velocities) and RGB-D images from virtual cameras on robot's head and wrist. The action space consists of four parts: a 1-dimensional discrete value indicating whether to extend the arm, 2-dimensional continuous values representing the robot base's linear and angular velocities, 6-dimensional continuous values representing the desired pose change of the end-effector, and 1-dimensional discrete value indicating whether to open the gripper.

Results

In our first experiment, training manipulation-only policies with imitation learning, we observe 98% success rate over 100 evaluation episodes. This experiment showcases that the human-iGibson interface enables easy collection of effective demonstrations for imitation learning, and the integrated motion planner is helpful for policy training. These two integrated features, as shown in Fig. 4.2, are novel combinations offered by iGibson. In the second experiment, training mobile-manipulation policies with imitation learning, we observe 70% success rate over 20 evaluation episodes. This experiment indicates that the we can leverage iGibson to train imitation learning algorithms for mobile manipulation tasks.

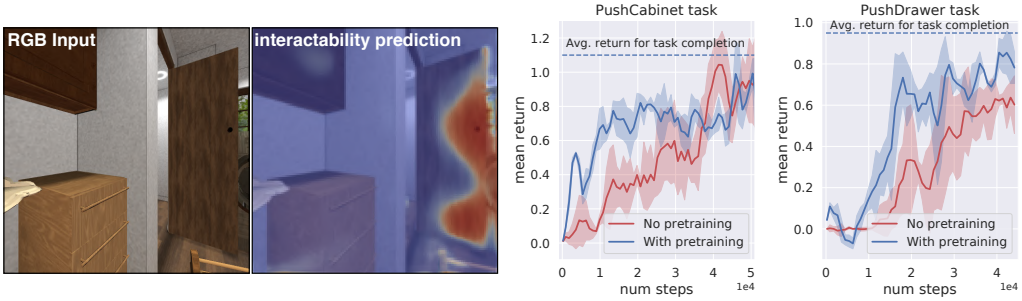


Figure 4.8: *Left:* Example result of interaction pretraining. The agent receives RGB input and predicts if the pixels are pushable (red: higher probability; blue: lower) (Sec. 4.4.3). The model learns to associate edge of doors as the most pushable points. *Right:* Training curves for two interactive tasks (`PushDrawer`,`PushCabinet`) with and without interaction pretraining.

4.4.3 Pretraining in Fully Interactive Scenes

In the third and final set of experiments, we evaluate the potential of using iGibson's fully interactive scenes to learn an intermediate visual representation that encodes the expected outcome of interactions with different objects. Such an intermediate visual representation may be used to accelerate robot learning of manipulation tasks, since they typically require the agent to associate visual observations with promising areas of interaction to change the state of the scene towards a manipulation goal.

To learn such representations, we set up a virtual agent that interacts with random points in the scenes and learns to predict the outcome of these interactions. The interaction is parameterized as a coordinate in the virtual agent's image observation space. We emulate a pushing interaction by displacing the corresponding 3D location of the selected pixel by 30 cm in the opposite direction of the surface normal, applying a maximum force of 60 N (a common payload of commercial robots). A motion of the point for more than 10 cm is considered a success. We sample 10 random pushes at each location, 4,000 locations in each scene. We use the images annotated with interaction successes/failures to train a U-Net [180]-based visual encoder that predicts heatmaps of expected interaction success from RGB input.

For the second phase, we train two policy networks for two manipulation task respectively (`PushDrawer, PushCabinet`). The goal is to close the drawers or the cabinets. The policy outputs points to interact (push) that are given to one of our integrated motion planners to generate an arm motion [240]. We use DQN [225] as policy learning algorithm. The predicted interaction heatmaps are used to gate the Q-value maps predicted by the network.

Results

Fig. 4.8 (left) depicts the result of the pretrained visual model. We observe that the heatmap has stronger activation at the edge of the door than in the area closer to the hinge, and weak activation on closed cabinets. The model learns to identify the best areas to push to cause motion (further visualizations on project website). For both downstream tasks, we observe that using the pre-trained representation significantly accelerates training (Fig. 4.8 (right)). This suggests that the full interactability of iGibson can help agents learn useful visual representation for downstream mobile manipulation tasks. Having a subset of objects not physically interactable will lead to false negatives during pretraining, and prevents successful representation learning. As shown in Fig. 4.2 and discussed in Sec. 4.2, fully interactive scenes with continuous robot actions is a specialty of iGibson.

4.5 Conclusion

We presented iGibson, a novel simulation environment for developing interactive robotic agents in large-scale realistic scenes. iGibson includes 15 fully interactive scenes with 108 rooms, and novel capabilities to generate high-quality virtual sensor signals, domain randomization, integration with motion planners, and human-iGibson interface. Through experiments, we showcased that iGibson helps to develop robust policies for navigation and manipulation. We hope that iGibson can aid researchers in solving complex robotics problems in large-scale realistic scenes.

Chapter 5

Visual Sim-to-Real Transfer

5.1 Introduction

Deep reinforcement learning (RL) from image observations has seen much success in various domains [141, 120, 4]. However, generalization remains a major obstacle towards reliable deployment. Recent studies have shown that RL agents struggle to generalize to new environments, even with similar tasks [54, 58, 35, 201]. This suggests that the learned RL policies fail to develop robust representations against irrelevant environmental variations.

Factors of variation in RL problems can be grouped into three main categories: generalization over different visual appearances [36, 58, 116], dynamics [156], and environment structures [230, 11, 35]. In this work, we mainly focus on zero-shot generalization to unseen environments of different visual appearances, but the same semantics.

One well-explored solution for better generalization is data augmentation [113]. For image observations in RL, augmentation can be either manually engineered into the simulator, also known as domain randomization [219], or automatic [109]. Prior works [12, 200] distinguish between *weak* augmentations like random cropping, and *strong* augmentations that heavily distort the image, such as Mixup [259] and Cutmix [254]. Strong augmentations are known to induce robust and generalizable representations for image classification [81]. However, naively transplanting them to RL hinders training and results in suboptimal performance [109]. Therefore, weak augmentations like random cropping are the most effective for RL at training time [106]. This poses a dilemma: more aggressive augmentations are necessary to cultivate better generalization for the visual domain [81], but RL does not benefit to the same extent as supervised learning since the training is fragile to excessive data variations.

We argue that the dilemma exists because it conflates two problems: policy learning and robust representation learning. To decouple them, we draw inspiration from policy distillation [184] where a student policy distills knowledge from one or more experts. The technique is used for different

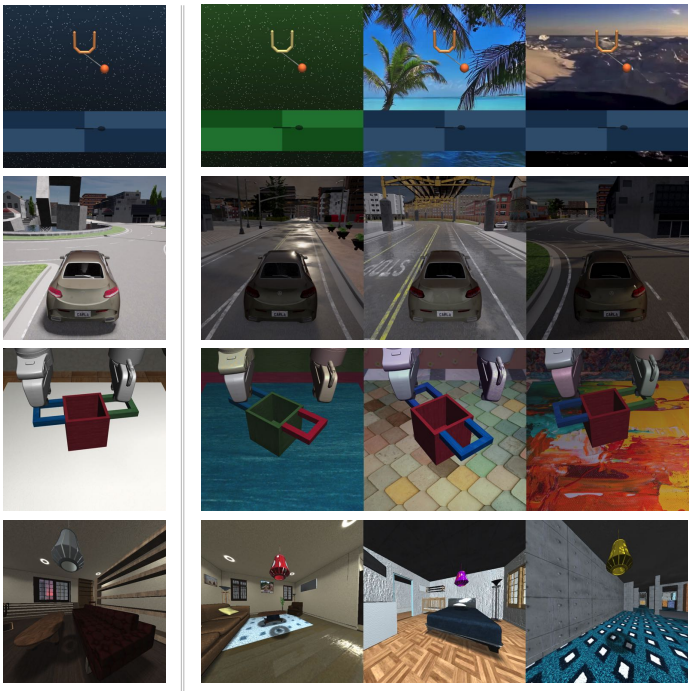


Figure 5.1: Our proposed benchmark for visual policy generalization in 4 diverse domains. Top to bottom: DMControl Suite (15 settings), CARLA autonomous driving (5 weathers), Robosuite (12 settings), and iGibson indoor navigation (20 rooms).

purposes, such as efficient policy deployment, multi-task RL, and policy transfer [216, 6, 39]. In this work, we introduce a new instantiation of policy distillation that addresses the dilemma effectively.

Our **key insight** is to solve policy optimization first, and then to robustify its representation by imitation learning with strong augmentations. First, an expert neural network is trained by RL with random cropping on the original environment. It learns a high-performance policy but cannot handle distribution shifts. Second, a student network learns to mimic the behavior of the expert, but with a crucial difference: the expert computes the ground-truth actions from *unmodified* observations, while the student learns to predict the same actions from heavily *corrupted* observations. The student optimizes a supervised learning objective, which has better training stability than RL, and the strong augmentations greatly remedy overfitting at the same time. Thus, SECANT is able to acquire robust representations without sacrificing policy performance.

Our method is strictly zero-shot, because no reward signal is allowed at test time, and neither the expert nor the student sees the test environments during training. SECANT is trained once and does not perform any test-time adaptation. In contrast, PAD [72], a prior SOTA method, adds a self-supervised auxiliary loss on intermediate representations during training, and continues to

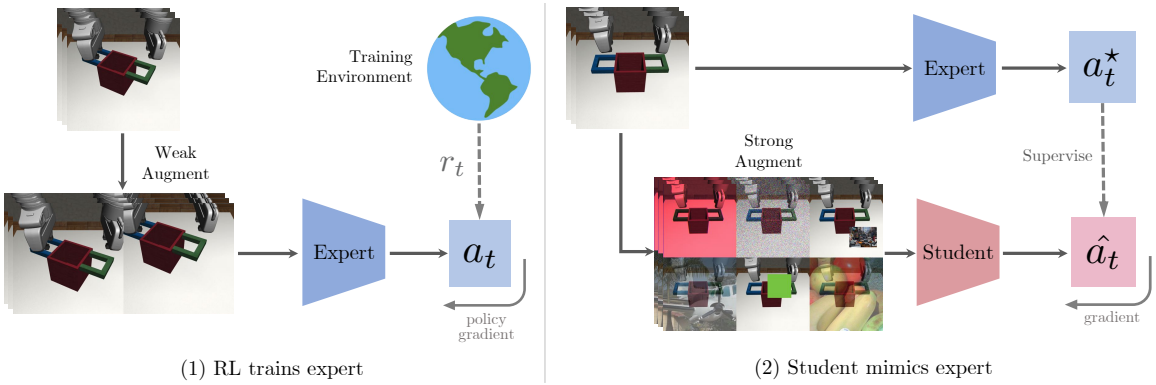


Figure 5.2: Algorithm overview. SECANT training is split into two stages. **Left, stage 1**: expert policy is trained by RL with weak augmentation (random cropping). **Right, stage 2**: student receives ground-truth action supervision from the expert at every time step, conditioned on the same observation but with strong augmentations, such as cutout-color, Gaussian noise, Mixup, and Cutmix. The student learns robust visual representations invariant to environment distractions, while maintaining high policy performance.

fine-tune the policy weights using this loss signal at testing. SECANT is more efficient compared to PAD, because the latter requires expensive gradient computation at every inference step and is impractical to deploy on mobile robots.

We benchmark on Deepmind Control Suite (DMControl) with randomized color and video backgrounds [72], and show that SECANT is able to outperform prior SOTA in **14 out of 15** unseen environments with an average score increase of **26.5%**. While DMControl is a popular benchmark, its test-time variations are artificial and not representative of real applications. Therefore, we further construct 3 new benchmarks with more realistic distribution shifts (Fig. 5.1), based on existing simulators: **(1) Robosuite** [266]: single-hand and bimanual robotic tasks. We add new appearances of table, background, and objects of interest that are varied at test time; **(2) CARLA** [46]: autonomous driving across 5 unseen weather conditions that feature highly realistic rendering of raining, sunlight changes, and shadow effects. **(3) iGibson** [195]: indoor object navigation in 20 distinct rooms with a large variety of interior design and layouts that we standardize. We hope that these new challenging environments will facilitate more progress towards generalizable visual policy learning.

To summarize our contributions:

- We propose SECANT (**S**elf **E**xpert **C**loning for **A**daptation to **N**ovel **T**est-environments), a novel algorithm that solves policy learning and robust representation learning *sequentially*, which achieves strong zero-shot generalization performance to unseen visual environments.

- We design and standardize a diverse and challenging suite of benchmarks in 4 domains: Deepmind Control Suite (DMControl), autonomous driving, robotic manipulation, and indoor object navigation. Except for DMControl, the other 3 environments feature test-time visual appearance drifts that are representative of real-world applications.
- We demonstrate that SECANT is able to dominate prior state-of-the-art methods on the majority of tasks, often by substantial margins, across all 4 domains.

5.2 Related Work

5.2.1 Generalization in Deep RL.

There is a plethora of literature that highlights the overfitting problem in deep RL [175, 156, 256, 99, 130, 36, 229, 35, 252, 174]. One class of approach is to re-design the training objectives to induce invariant representations directly. [257] and [203] aim to learn robust features via deep metric learning [56]. [177] combines RL with CycleGAN. [98] employs automatic curriculum for generalization. PAD [72] adds a self-supervised auxiliary component that can be adapted at test time. In contrast to these prior works, SECANT is a plug-and-play method that neither modifies the existing RL algorithm, nor requires computationally expensive test-time fine-tuning. Similar to us, ATC [206] separates representation learning from RL. It pretrains an encoder, fine-tunes with reward, and evaluates in the *same* environment. In contrast, SECANT solves policy learning first before robustification, and focuses heavily on zero-shot generalization instead.

Other works [54, 36] apply regularization techniques originally developed for supervised learning, such as L2 regularization, BatchNorm [91], and dropout [204]. [89] regularizes RL agents via selective noise injection and information bottleneck. These methods improve policy generalization in Atari games and CoinRun [36]. SECANT is orthogonal to these techniques and can be combined for further improvements. We also contribute a new benchmark with more realistic tasks and variations than video games.

5.2.2 Data Augmentation and Robustness

Semantic-preserving image transformations have been widely adopted to improve the performance and robustness of computer vision systems [80, 81, 13, 200]. Domain randomization (DR) [219, 162] produces randomized textures of environmental components. It is a special type of data augmentation that requires extensive manual engineering and tuning of the simulator [165, 250]. RL training, however, benefits the most from weak forms of augmentations that do not add extra difficulty to the policy optimization process [109, 106, 173, 72]. By design, SECANT unlocks a multitude of strong augmentation operators that are otherwise suboptimal for training in prior works. We successfully

employ techniques from supervised image classification like Cutmix [254] and Mixup [259]; the latter has also been explored in [231].

5.2.3 Policy Distillation

SECANT belongs to the policy distillation family, a special form of knowledge distillation [82] for RL. Prior works use policy distillation for different purposes [39]. [26] and [115] train an expert with privileged simulator information (e.g. groundtruth physical states) to supervise a student policy that can only access limited sensors at deployment. [263] transfers navigation policies across domains through an intermediate proxy model. [90] reduces the non-stationary effects of RL environment by repeated knowledge transfer. Other works involve multi-task student networks [184, 216, 6] that distill from multiple experts simultaneously. SECANT differs from these works because our expert and student share the same task and observation information, but shoulder different responsibilities: expert handles policy optimization while student addresses visual generalization.

Our method is related to FixMatch [200], which imposes a pseudo-label distillation loss on two different augmentations of the same image. In contrast, our expert only needs to overfit to the training environment, while the student distills from a frozen expert to learn robust representation. Concurrent work [73] also validates the benefit of decoupling and strong augmentation. In comparison, SECANT is conceptually simpler and does not require modifying the RL training pipeline. Another closely related field is imitation learning [190, 5, 181, 84]. Our student imitates without external demonstration data, hence the name "self-expert cloning".

5.3 Background

5.3.1 Soft Actor-Critic

In this work, we mainly consider continuous control from raw pixels. The agent receives an image observation $o \in \mathbb{R}^{C\times H\times W}$ and outputs a continuous action $a \in \mathbb{R}^d$. SAC [68, 69] is a state-of-the-art off-policy RL algorithm.

It learns a policy $\pi(a|o)$ and a critic $Q(o,a)$ that maximize a weighted combination of reward and policy entropy, $\mathbb{E}_{(o_t,a_t)\sim\pi}\left[\sum_t r_t + \alpha\mathcal{H}(\pi(\cdot|o_t))\right]$. SAC stores experiences into a replay buffer $\mathcal{D}$. The critic parameters are updated by minimizing the Bellman error using transitions sampled from $\mathcal{D}$:

$$\mathcal{L}_Q = \mathbb{E}_{(o_t,a_t)\sim\mathcal{D}}\left[\Big(Q(o_t,a_t) - (r_t + \gamma V(o_{t+1}))\Big)^2\right] \tag{5.1}$$

By sampling an action under the current policy, we can estimate the soft state value as following:

$$V(o_{t+1}) = \mathbb{E}_{a'\sim\pi}\left[\bar{Q}(o_{t+1},a') - \alpha\log\pi(a'|o_{t+1})\right] \tag{5.2}$$

where $\bar{Q}$ denotes an exponential moving average of the critic network.

The policy is updated by minimizing the divergence from the exponential of the soft-Q function:

$$\mathcal{L}_\pi = -\mathbb{E}_{a_t \sim \pi} \left[Q(o_t, a_t) - \alpha \log \pi(a_t | o_t) \right] \tag{5.3}$$

where α is a learnable temperature parameter that controls the stochasticity of the optimal policy.

5.3.2 Dataset Aggregation

DAgger [181] is an iterative imitation learning algorithm with strong performance guarantees. First, it rolls out an expert policy π_e to seed an experience dataset $\mathcal{D}^0$. The student policy π_s^0 is trained by supervised learning to best mimic the expert on those trajectories. Then at iteration i, it rolls out π_s^i to collect more trajectories that will be added to $\mathcal{D}^i$. π_s^{i+1} will then be trained on the new *aggregated* dataset $\mathcal{D}^{i+1}$, and the process repeats until convergence. Even though more advanced imitation algorithms have been developed [84], DAgger is conceptually simple and well-suited for SECANT because our student network can query the expert for dense supervision at every time step.

5.4 Secant

The goal of our proposed self-expert cloning technique is to learn a robust policy that can generalize zero-shot to unseen visual variations. SECANT training can be decomposed into two stages. Algorithm 1 shows the full pseudocode.

5.4.1 Expert Policy

In the first stage, we train a high-performance expert policy in the original environment with weak augmentations. In visual continuous control tasks, the policy is parametrized by a feed-forward deep convolutional network $\pi_e(\mathcal{O}; \theta_e) : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^d$ that maps an image observation to a d-dimensional continuous action vector. In practice, we employ a frame stacking technique that concatenates T consecutive image observations along the channel dimension to incorporate temporal information [141]. The augmentation operator is a semantic-preserving image transformation $f : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^{C' \times H' \times W'}$. Prior works have found that random cropping performs the best in a range of environments, therefore we adopt it as the default weak augmentation for the expert [109].

The expert can be optimized by any standard RL algorithm. We select Soft Actor-Critic (SAC) due to its wide adoption in continuous control tasks [68, 69]. The expert is optimized by gradient descent to minimize the SAC objectives (Equations 5.1 and 5.3). Since we place little restrictions on the expert, our method can even be used to robustify pre-trained policy network checkpoints, such as the RL Model Zoo [172].

Algorithm 1 SECANT: Self-Expert Cloning

π_e, π_s: randomly initialized expert and student policies
$\mathcal{F}_{weak}$, $\mathcal{F}_{strong}$: sets of image augmentations
$\mathcal{B}$: experience replay buffer
for t in $1, \ldots, T_{RL}$ **do**
  Sample experience batch $\tau_t = (o_t, a_t, o_{t+1}, r) \sim \mathcal{B}$
  Sample weak augmentation $f \sim \mathcal{F}_{weak}$
  Augment $o_t = f(o_t); o_{t+1} = f(o_{t+1})$
  Update π_e to minimize $\mathcal{L}_{RL}(\tau_t)$
end for
Roll out π_e to collect an initial dataset $\mathcal{D}$ of trajectories
for t in $1, \ldots, T_{imitate}$ **do**
  Sample observation batch $o \sim \mathcal{D}$
  Sample strong augmentation $f \sim \mathcal{F}_{strong}$
  Update π_s to minimize $\|\pi_s(f(o)) - \pi_e(o)\|_F$
  Roll out π_s for one environment step and add to the dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{o_s\}$
end for

5.4.2 Student Policy Distillation

In the second stage, we train a student network to predict the optimal actions taken by the expert, conditioned on the same observation but with heavy image corruption. This stage does *not* need further access to the reward signal. Formally, the student is also a deep convolutional network $\pi_s(\mathcal{O}; \theta_s) : \mathbb{R}^{C \times H \times W} \rightarrow \mathbb{R}^d$ that may have different architecture from the expert. The student policy distills from the expert following the DAgger imitation procedure (Sec 5.3). First, we roll out the *expert* policy to collect an initial dataset $\mathcal{D}$ of trajectories. Next, at each iteration, we select a strong augmentation operator $f \sim \mathcal{F}_{strong}$ and apply it to a batch of observations o sampled from $\mathcal{D}$. We alternate between (1) updating the student's parameters by gradient descent on a supervised regression loss: $\mathcal{L}(o; \theta_s) = \|\pi_s(f(o)) - \pi_e(o)\|_F$ and (2) adding more experiences to $\mathcal{D}$ under the latest *student* policy.

In the experiments, we consider 1 type of weak augmentation (random cropping) and 6 types of strong augmentation techniques developed in RL and robust image classification literature [80, 81, 109, 116]. We refer to weak augmentations as the ones that can substantially improve RL optimization at training time, while strong augmentations are less effective as they make training more difficult. We focus only on random cropping for weak augmentation in this work, and defer other potential operators to future works. Below are brief descriptions of the augmentations we study:

- **Cutout-color (`Cc`)**: inserts a small rectangular patch of random color into the observation at a random position.
- **Random convolution (`Cv`)**: passes the input observation through a random convolutional layer.

- **Gaussian (`G`)**: adds Gaussian noise.
- **Impulse (`I`)**: adds the color analogue of salt-and-pepper noise.
- **Mixup (`M`)** [259]: linearly blends the observation with a distracting image I: $f(o) = \alpha o + (1 - \alpha)I$. We randomly sample I from 50K COCO images [128], and sample $\alpha \sim \text{Uniform}(0.2, 0.6)$.
- **Cutmix (`Cm`)** [254]: similar to Cutout-color except that the patch is randomly sampled from COCO images.

These augmentations can be categorized into low-frequency noise (`Cc` and `Cv`), high-frequency unstructured noise (`G` and `I`), and high-frequency structured noise (`M` and `Cm`). Mixup and Cutmix with image distractions are novel operators that have not been studied for visual policy generalization.

We also investigate combinations of the above, and find empirically that random sampling from low-frequency and high-frequency *structured* noise types yields the best overall results. We note that adding random cropping to the mix benefits performance slightly, likely because it improves the spatial invariance of the student's representation. We design three combination recipes (Table 5.2): Combo1 (`Cc+Cv+M+Crop`), Combo2 (`Cc+Cv+M+Cm+Crop`), and Combo3 (`Cc+Cv+M`). We have not done an exhaustive search, so it is possible that better combinations exist.

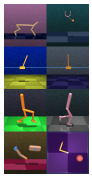
5.5 Experiments

We propose a new benchmark of 4 diverse domains (Fig. 5.1) to systematically assess the generalization ability of visual agents. They offer a wide spectrum of visual distribution shifts for testing. In each domain, we investigate how well an algorithm trained in one environment performs on various unseen environments in a zero-shot setting, which disallows reward signal and extra trials at test time.

For each task, we benchmark SECANT extensively against prior state-of-the-art algorithms:

- **SAC**: plain SAC with no augmentation.
- **SAC+crop**: SAC with time-consistent random cropping [106].
- **DR**: domain randomization. To simulate realistic deployment, our randomized training distributions are narrower than the test distributions.
- **NetRand**: Network Randomization [116], which augments the observation image by random convolution.
- **SAC+IDM**: SAC trained with an auxiliary inverse dynamics loss [160].
- **PAD**: prior SOTA method on top of SAC+IDM that fine-tunes the auxiliary head at test time [72].

Table 5.1: DMControl: SECANT outperforms prior SOTA methods substantially in **14 out of 15** settings with **+26.5%** boost on average.

Setting	Task	SECANT (Ours)	SAC	SAC+crop	DR	NetRand	SAC+IDM	PAD
	Cheetah run	**582 ± 64** (+88.3%)	133 ± 26	100 ± 27	145 ± 29	309 ± 66	121 ± 38	159 ± 28
	Ball in cup catch	**958 ± 7** (+ 8.1%)	151 ± 36	359 ± 76	470 ± 252	886 ± 57	471 ± 75	563 ± 50
	Cartpole swingup	**866 ± 15** (+27.2%)	248 ± 24	537 ± 98	647 ± 48	681 ± 122	585 ± 73	630 ± 63
	Cartpole balance	**992 ± 6** (+ 0.8%)	930 ± 36	769 ± 63	867 ± 37	984 ± 13	835 ± 40	848 ± 29
	Walker walk	**856 ± 31** (+27.6%)	144 ± 19	191 ± 33	594 ± 104	671 ± 69	406 ± 29	468 ± 47
	Walker stand	**939 ± 7** (+ 4.3%)	365 ± 79	748 ± 60	715 ± 96	900 ± 75	743 ± 37	797 ± 46
	Finger spin	**910 ± 115** (+ 3.1%)	504 ± 114	847 ± 116	465 ± 314	883 ± 156	757 ± 62	803 ± 72
	Reacher easy	**639 ± 63** (+29.1%)	185 ± 70	231 ± 79	105 ± 37	495 ± 101	201 ± 32	214 ± 44
	Cheetah run	**428 ± 70** (+56.8%)	80 ± 19	102 ± 30	150 ± 34	273 ± 26	164 ± 42	206 ± 34
	Ball in cup catch	**903 ± 49** (+57.3%)	172 ± 46	477 ± 40	271 ± 189	574 ± 82	362 ± 69	436 ± 55
	Cartpole swingup	**752 ± 38** (+44.3%)	204 ± 20	442 ± 74	485 ± 67	445 ± 50	487 ± 90	521 ± 76
	Cartpole balance	**863 ± 32** (+12.7%)	569 ± 79	641 ± 37	766 ± 92	708 ± 28	691 ± 76	687 ± 58
	Walker walk	**842 ± 47** (+17.4%)	104 ± 14	244 ± 83	655 ± 55	503 ± 55	694 ± 85	717 ± 79
	Walker stand	932 ± 15	274 ± 39	601 ± 36	869 ± 60	769 ± 78	902 ± 51	**935 ± 20**
	Finger spin	**861 ± 102** (+21.6%)	276 ± 81	425 ± 69	338 ± 207	708 ± 170	605 ± 61	691 ± 80

Following prior works [72] on DMControl, we repeat training across 10 random seeds to report the mean and standard deviation of the rewards. We use 5 random seeds for all other simulators and ablation studies.

5.5.1 Algorithm Details

SECANT builds upon SAC, and adopts similar hyperparameters and network architecture as [106]. Observations are stacks of 3 consecutive RGB frames. For all tasks, we use a 4-layer feed-forward ConvNet with no residual connection as encoder for both the SECANT expert and student, although they do not have to be identical. PAD, however, requires a deeper encoder network (11 layers) to perform well in DMControl [72]. For all other simulators, we conduct a small grid search and find that 6-layer encoders work best for both SAC+IDM and PAD. After the encoder, 3 additional fully connected layers map the visual embedding to action.

We implement Soft Actor Critic [68, 69] for all environments in PyTorch v1.7 [159] with GPU acceleration for visual observations. We follow the random cropping scheme in [106], which first applies a reflection padding to expand the image, and then crop back to the original size. Some of the augmentation operators are implemented with the Kornia library [179].

5.5.2 Deepmind Control Suite

We follow the settings in [72] and experiment with 8 tasks from DMControl. We measure generalization to (1) randomized colors of the background and robot itself, and (2) natural videos as dynamic background (Fig. 5.1). SECANT *significantly outperforms* prior SOTA in all but one task, often by substantial margins up to **88.3%** (Table 5.1). All methods are trained for 500K steps with dense task-specific rewards. SAC+crop is the same as SECANT's expert, from which the student distills

Table 5.2: Ablation on student augmentations: given the same experts trained with random cropping, we ablate 6 strong augmentations and their mixtures for the student. Combo[1-3] randomly select an augmentation from their pool to apply to each observation.

Setting	Tasks	Combo1	Combo2	Combo3	Cutout-color	Conv	Mixup	Cutmix	Gaussian	Impulse
Color	Cartpole	**866 ± 15**	865 ± 17	863 ± 15	776 ± 33	860 ± 15	825 ± 16	751 ± 43	720 ± 86	751 ± 45
	Cheetah	**582 ± 64**	522 ± 166	570 ± 50	343 ± 153	318 ± 123	222 ± 38	303 ± 82	373 ± 110	382 ± 121
	Walker	856 ± 31	854 ± 27	832 ± 45	701 ± 75	**866 ± 22**	756 ± 46	658 ± 54	770 ± 56	727 ± 47
Video	Cartpole	752 ± 38	765 ± 55	**778 ± 37**	607 ± 31	556 ± 61	677 ± 43	647 ± 56	580 ± 61	639 ± 22
	Cheetah	**428 ± 70**	409 ± 31	406 ± 33	183 ± 46	229 ± 30	309 ± 65	209 ± 43	196 ± 38	224 ± 25
	Walker	**842 ± 47**	836 ± 36	792 ± 59	631 ± 52	531 ± 54	759 ± 64	675 ± 22	488 ± 31	471 ± 17
Example Augmentations										

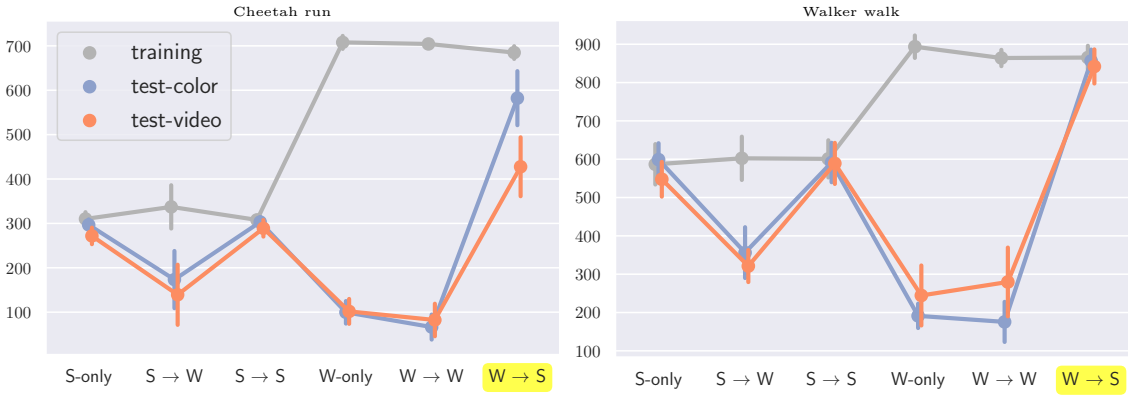


Figure 5.3: Ablation on different strategies to apply augmentation. "S-only" denotes single-stage policy trained with strong augmentation, and S → W means strongly-augmented expert imitated by weakly-augmented student. The recipe for SECANT is W → S.

for up to 30K steps without reward. SAC+IDM and PAD numbers are from [72].

Choice of student augmentations

We hypothesize that SECANT needs multiple kinds of augmentations to resist a wide variety of distribution shifts at test time. Keeping the experts fixed, we study the effect of 6 different augmentations and their combinations on the student (Table 5.2). In the challenging random video environments, Mixup and Cutmix tend to outperform other single operators. In most tasks, sampling from a mixture of augmentations generalizes better than solos, thus confirming our hypothesis. We adopt Combo1 for all SECANT results in Table 5.1.

Table 5.3: Ablation on imitation strategies. DAgger outperforms Expert-only and Student-only data collection in the second stage.

Setting	Task	DAgger	Expert	Student
Random Color	Cheetah run	**582 ± 64**	519 ± 73	347 ± 326
	Walker walk	**856 ± 31**	818 ± 41	854 ± 33
Random Video	Cheetah run	**428 ± 70**	291 ± 41	264 ± 241
	Walker walk	**842 ± 47**	778 ± 67	822 ± 55

Single stage vs two-stage augmentation

The premise of SECANT is that we cannot effectively apply strong augmentation (Combo1) in one stage to learn robust policies. We put this assumption to test. In Fig. 5.3, "S-only" and "W-only" are single-stage policies trained with strong or weak augmentations. $X \to Y$ denotes two-stage training, e.g. S $\to$ W means a strongly-augmented expert is trained first, and then a weakly-augmented student imitates it. We highlight **4 key findings**:

1. Single-stage RL trained with strong augmentation (S-only) underperforms in both training and test environments consistently, due to poor optimization.
2. The student is typically upper-bounded by the expert's performance, thus both S $\to$ W and S $\to$ S produce sub-optimal policies.
3. Single-stage policy trained with random cropping (W-only) overfits on the training environment and generalizes poorly. Adding a weakly-augmented student (W $\to$ W) does not remedy the overfitting.
4. The only effective strategy is a weakly-augmented expert followed by a strongly-augmented student (W $\to$ S), which is exactly SECANT. It recovers the strong performance on the training environment, and bridges the generalization gap in unseen test environments.

Ablation on imitation strategies

SECANT uses DAgger (Sec. 5.3) in the second stage, which rolls out the expert policy to collect initial trajectories, and then follows the student's policy. The alternatives are using expert or student policy alone to collect all trajectory data. The former approach lacks data diversity, while the latter slows down learning due to random actions in the beginning. Table 5.3 validates the choice of DAgger for policy distillation.

Ablation on the parallel-distillation variant.

Can we train the expert and the student at the same time, rather than sequentially? We consider a variant of our method, called SECANT-Parallel, that trains the expert alongside the student while

Table 5.4: Ablation on SECANT-Parallel variant. It is advantageous to train expert and student sequentially rather than in parallel.

Setting	Task	SECANT	SECANT-Parallel
Random Color	Cheetah run	**582 ± 64**	302 ± 248
	Ball in cup catch	**958 ± 7**	790 ± 332
	Cartpole swingup	**866 ± 15**	834 ± 8
	Walker walk	**856 ± 31**	768 ± 22
Random Video	Cheetah run	**428 ± 70**	276 ± 216
	Ball in cup catch	**903 ± 49**	676 ± 280
	Cartpole swingup	752 ± 38	**764 ± 17**
	Walker walk	**842 ± 47**	699 ± 21

Table 5.5: Robosuite results. The 3 sets of test environments are progressively harder (*easy*, *hard*, and *extreme*) with more distracting textures of the table, floor, and objects. SECANT gains an average of +337.8% reward over prior SOTA.

Setting	Tasks	SECANT (Ours)	SAC	SAC+crop	DR	NetRand	SAC+IDM	PAD
	Door opening	**782 ± 93** (+ 78.5%)	17 ± 12	10 ± 8	177 ± 163	438 ± 157	3 ± 2	2 ± 1
	Nut assembly	**419 ± 63** (+ 73.1%)	3 ± 2	6 ± 5	12 ± 7	242 ± 28	13 ± 12	11 ± 10
	Two-arm lifting	**610 ± 28** (+883.9%)	29 ± 11	23 ± 10	41 ± 9	62 ± 43	20 ± 8	22 ± 7
	Peg-in-hole	**837 ± 42** (+114.6%)	186 ± 62	134 ± 72	139 ± 37	390 ± 68	150 ± 41	142 ± 37
	Door opening	**522 ± 131** (+292.5%)	11 ± 10	11 ± 7	37 ± 31	133 ± 82	2 ± 1	2 ± 1
	Nut assembly	**437 ± 102** (+141.4%)	6 ± 7	9 ± 8	33 ± 18	181 ± 53	34 ± 28	24 ± 26
	Two-arm lifting	**624 ± 40** (+923.0%)	28 ± 11	27 ± 9	61 ± 15	41 ± 25	17 ± 6	19 ± 8
	Peg-in-hole	**774 ± 76** (+140.4%)	204 ± 81	143 ± 62	194 ± 41	322 ± 72	165 ± 75	164 ± 69
	Door opening	**309 ± 147** (+120.7%)	11 ± 10	6 ± 4	52 ± 46	140 ± 107	2 ± 1	2 ± 1
	Nut assembly	**138 ± 56** (+ 53.3%)	2 ± 1	10 ± 7	12 ± 7	90 ± 61	4 ± 3	4 ± 3
	Two-arm lifting	**377 ± 37** (+1156.7%)	25 ± 7	12 ± 6	30 ± 13	12 ± 11	24 ± 10	21 ± 10
	Peg-in-hole	**520 ± 47** (+ 75.7%)	164 ± 63	130 ± 81	154 ± 34	296 ± 90	155 ± 73	154 ± 72

keeping all other settings fixed. Similar to SECANT, it also enjoys the nice property of disentangling robust representation learning from policy optimization. However, the student in SECANT distills from a fully-trained and frozen expert, while the student in SECANT-Parallel has to distill from a non-stationary expert, which leads to suboptimal performances. Table 5.4 demonstrates that it is more beneficial to adopt our proposed two-stage procedure, as SECANT outperforms SECANT-Parallel in a majority of tasks.

5.5.3 Robosuite: Robotic Manipulation

Robosuite [266] is a modular simulator for robotic research. We benchmark SECANT and prior methods on 4 challenging single-arm and bimanual manipulation tasks.

We use the Franka Panda robot model with operational space control for 4 Robosuite tasks, and train with task-specific dense reward. All agents receive a 168×168 egocentric RGB view as input (example in Table 5.5). The action dimensions for door opening, nut assembly, peg-in-hole, and two-arm lifting are 7, 7, 12 and 14, respectively. We use continuous action space. The control frequency

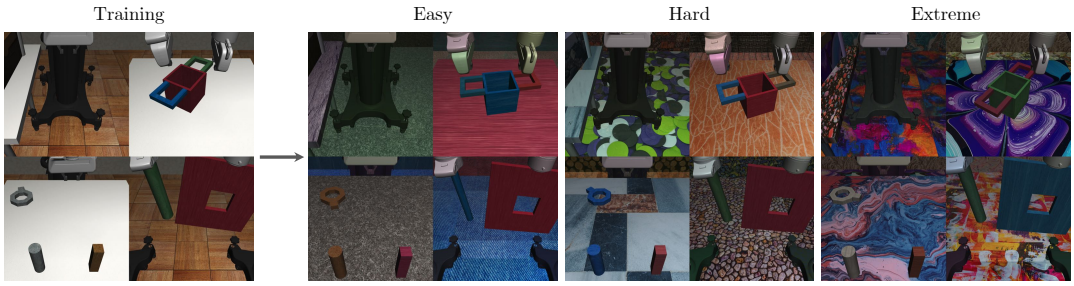


Figure 5.4: Sample Robosuite environments. Tasks in clockwise order: Door opening, Two-arm lifting, Peg-in-hole, and Nut assembly.

is set to 20 Hz, which means that the robot receives 20 control signals during every simulation second. The positions of moveable objects are randomized in each episode.

We provide brief descriptions of each task and their associated reward functions below. All environments add an extra positive reward upon task completion, in addition to the dense reward shaping. Example observations are shown in Fig. 5.4. Please refer to [266] for more details.

1. **Door opening.** A robot arm must learn to turn the handle and open the door in front of it. The reward is shaped by the distance between the door handle and the robot arm, and the rotation angle of the door handle.

2. **Nut assembly.** Two colored pegs (one square and one round) are mounted on the tabletop. The robot must fit the round nut onto the round peg. At first, the robot receives a reaching reward inversely proportional to the distance between the gripper and the nut, and a binary reward once it grasps the nut successfully. After grasping, it obtains a lifting reward proportional to the height of the nut, and a hovering reward inversely proportional to the distance between the nut and the round peg.

3. **Peg-In-Hole.** One arm holds a board with a square hole in the center, and the other holds a long peg. The two arms must coordinate to insert the peg into the hole. The reward is shaped by the distance between two arms, along with the orientation and distance between the peg and the hole.

4. **Two-arm lifting.** A large pot with two handles is placed on the table. Two arms on opposite ends must each grab a handle and lift the pot together above certain height, while keeping it level. At first, the agent obtains a reaching reward inversely proportional to the distance between each arm and its respective pot handle, and a binary reward if each gripper is grasping the correct handle. After grasping, the agent receives a lifting reward proportional to the pot's height above the table and capped at a certain threshold.

All agents are trained with clean background and objects, and evaluated on 3 progressively

Table 5.6: Robustness analysis in Robosuite: we measure the cycle consistency of observation embeddings across trajectories of the same task but different appearances. The higher the accuracy, the more robust the representation is against visual variations.

Cycle	Tasks	SECANT (Ours)	SAC	SAC+crop	DR	NetRand	SAC+IDM	PAD
2-way	Nut assembly	**77.3 ± 7.6** (+29.3%)	24.0 ± 6.0	16.0 ± 3.7	25.3 ± 11.9	48.0 ± 15.9	29.3 ± 13.0	26.7 ± 9.4
	Two arm lifting	**72.0 ± 9.9** (+32.0%)	20.0 ± 0.0	18.7 ± 3.0	24.0 ± 10.1	40.0 ± 14.9	18.7 ± 3.0	18.7 ± 5.6
3-way	Nut assembly	**33.3 ± 8.2** (+17.3%)	16.0 ± 8.9	16.0 ± 3.7	8.0 ± 11.0	9.3 ± 10.1	10.7 ± 7.6	10.7 ± 7.6
	Two arm lifting	**32.0 ± 8.7** (+20.0%)	6.7 ± 9.4	2.7 ± 3.7	10.7 ± 10.1	6.7 ± 6.7	12.0 ± 7.3	12.0 ± 7.3

harder sets of environments (Table 5.5). We design 10 variations for each task and difficulty level, and report the mean reward over 100 evaluation episodes (10 per variation). Reward below 100 indicates a failure to solve the task.

SECANT gains an average of **+287.5%** more reward in *easy* set, **+374.3%** in *hard* set, and **+351.6%** in *extreme* set over the best prior method. The *hard* and *extreme* settings are particularly challenging because the objects of interest are difficult to discern from the noisy background. For nut assembly and two-arm lifting, SECANT is the only agent able to obtain non-trivial partial rewards in *hard* and *extreme* modes consistently.

Embedding robustness analysis

To verify that our method develops high-quality representation, we measure the cycle consistency metric proposed in [8]. First, given two trajectories U and V, observation $u_i \in U$ locates its nearest neighbor in V: $v_j = \arg\min_{v \in V} \|\phi(u_i) - \phi(v)\|^2$, where $\phi(\cdot)$ denotes the 50-D embedding from the visual encoder of the learned policies. Then in reverse, v_j finds its nearest neighbor from U: $u_k = \arg\min_{u \in U} \|\phi(u) - \phi(v_j)\|^2$. u_i is cycle consistent if and only if $|i - k| \leq 1$, i.e. it returns to the original position. High cycle consistency indicates that the two trajectories are accurately aligned in the embedding space, despite their visual appearance shifts.

We also evaluate 3-way cycle consistency that involves a third trajectory W, and measure whether u_i can return to itself along *both* $U \to V \to W \to U$ and $U \to W \to V \to U$. In Table 5.6, we sample 15 observations from each trajectory, and report the mean cycle consistency over 5 trials. In Fig. 5.5, we also visualize the state embeddings of door-opening task with t-SNE [129].

Both quantitative and qualitative analyses show that SECANT significantly improves the robustness of visual representation over the baselines.

Saliency visualization

To better understand how the agents execute their policies, we compute saliency maps as described in [64]. We add Gaussian perturbation to the observation image at every 5×5 pixel patch, and visualize the saliency patterns in Fig. 5.5. SECANT is able to focus on the most task-relevant objects, even with novel textures it has not encountered during training.

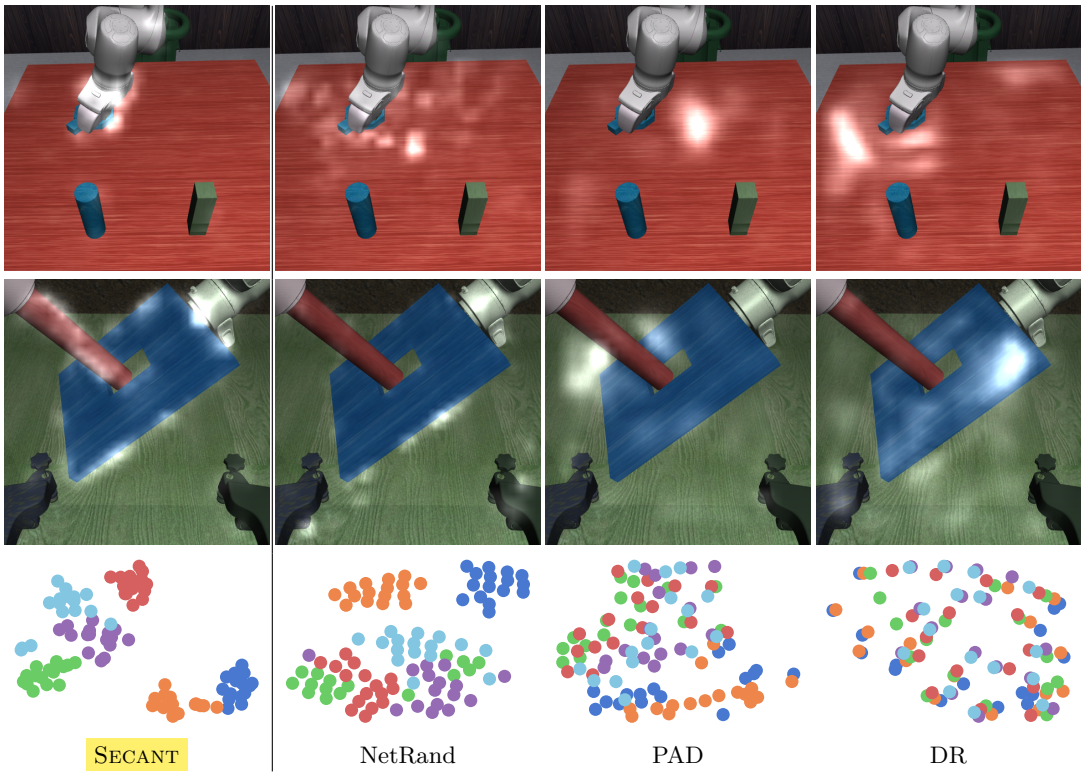


Figure 5.5: Row 1 and 2: saliency map of the learned policies in unseen tests. SECANT attends to the components crucial to the task, while other agents often focus on irrelevant places. Row 3: t-SNE visualization of state embeddings. Our method correctly groups semantically similar states with different visual appearances.

5.5.4 CARLA: Autonomous Driving

To further validate SECANT's generalization ability on natural variations, we construct a realistic driving scenario with visual observations in the CARLA simulator [46]. The goal is to drive as far as possible along a figure-8 highway (CARLA Town 4) in 1000 time steps without colliding into 60 moving pedestrians or vehicles. Our reward function is similar to [258], which rewards progression, penalizes collisions, and discourages abrupt steering. The RGB observation is a 300-degree panorama of 84×420 pixels, formed by concatenating 5 cameras on the vehicle's roof with 60-degree view each. The output action is a 2D vector of thrust (brake is negative thrust) and steering.

We implement the environment in CARLA v0.9.9.4 [46] and adopt the reward function in [258]:

$$r_t = \mathbf{v}_{\text{agent}}^{\top} \hat{\mathbf{u}}_{\text{highway}} \cdot \Delta t - \lambda_c \cdot \text{collision} - \lambda_s \cdot |\text{steer}| \tag{5.4}$$

where $\mathbf{v}_{\text{agent}}$ is the velocity vector of our vehicle, and the dot product with the highway's unit vector $\hat{\mathbf{u}}_{\text{highway}}$ encourages progression along the highway as fast as possible. $\Delta t = 0.05$ discretizes

Table 5.7: CARLA autonomous driving. The different weathers feature highly realistic raining, shadow, and sunlight changes. We report distance (m) travelled in a town without collision. SECANT drives **+47.7%** farther on average than other agents at test time.

Setting	Weather	SECANT (Ours)	SAC	SAC+crop	DR	NetRand	SAC+IDM	PAD
Training	Clear noon	596 ± 77	282 ± 71	**684 ± 114**	486 ± 141	648 ± 61	582 ± 96	632 ± 126
Test Weathers	Wet sunset	**397 ± 99** (+ 39.8%)	57 ± 14	26 ± 18	9 ± 11	284 ± 84	25 ± 11	36 ± 12
	Wet cloudy noon	**629 ± 204** (+ 5.7%)	180 ± 45	283 ± 85	595 ± 260	557 ± 107	433 ± 105	515 ± 52
	Soft rain sunset	**435 ± 66** (+ 73.3%)	55 ± 28	38 ± 25	25 ± 41	251 ± 104	36 ± 32	41 ± 37
	Mid rain sunset	**470 ± 202** (+101.7%)	50 ± 8	37 ± 16	24 ± 24	233 ± 117	42 ± 23	32 ± 21
	Hard rain noon	**541 ± 96** (+ 18.1%)	237 ± 85	235 ± 129	341 ± 96	458 ± 72	156 ± 194	308 ± 141

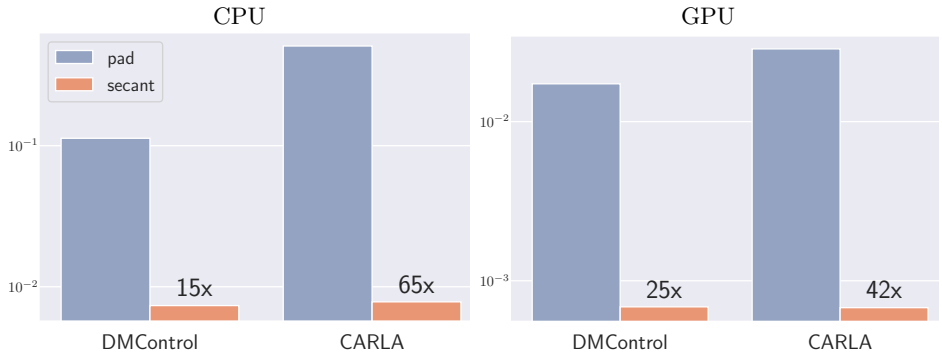


Figure 5.6: SECANT vs PAD inference latency. Y-axis denotes average seconds per action (log-scale). SECANT improves inference speed by an order of magnitude compared to PAD.

the simulation time. We penalize collision, measured as impulse in Newton-seconds, and excessive steering. The respective coefficients are $\lambda_c = 10^{-4}$ and $\lambda_s = 1$. We do not investigate more sophisticated rewards like lane-keeping and traffic sign compliance, as they are not the main focus of this paper. We use action repeat 8 for all agents.

The agents are trained at "clear noon", and evaluated on a variety of dynamic weather and lighting conditions at noon and sunset (Fig. 5.1). For instance, the *wet* weathers feature roads with highly reflective spots. Averaged over 10 episodes per weather and 5 training runs, SECANT is able to drive **+47.7%** farther than prior SOTAs in tests.

Inference speed

At inference time, latency between observing and acting is crucial for real-world deployment. Unlike SECANT that only performs a single forward pass, PAD [72] requires expensive test-time gradient computation. In addition, PAD needs a deeper ConvNet as encoder and extra test-time image augmentations for the auxiliary self-supervised mechanism to work well [72]. These add even more overhead during inference.

We benchmark both SECANT and PAD on actual hardware for DMControl and CARLA (Fig. 5.6).

Table 5.8: iGibson object navigation. The goal is to find and navigate to a ceiling lamp in unseen rooms with novel decoration, furniture, and layouts (sample floor plan below). In testing, SECANT has **+15.8%** higher success rate (absolute percentage) than competing methods.

Setting	SECANT (Ours)	SAC	SAC+crop	DR	NetRand	SAC+IDM	PAD
Training	64.0 ± 3.7	$\mathbf{68.7 \pm 2.5}$	51.0 ± 12.0	49.6 ± 12.7	56.4 ± 3.8	54.2 ± 8.8	59.0 ± 13.4
Test: Easy	$\mathbf{56.8 \pm 17.2}$ (+17.6%)	13.8 ± 7.5	12.9 ± 7.1	17.6 ± 13.2	39.2 ± 11.7	25.9 ± 12.4	30.9 ± 12.4
Test: Hard	$\mathbf{47.7 \pm 11.3}$ (+13.9%)	9.3 ± 7.6	7.9 ± 5.3	15.2 ± 15.3	33.8 ± 11.8	12.7 ± 8.3	26.1 ± 23.0

The CPU model is Intel Xeon Gold 5220 (2.2 GHz) CPU, and the GPU model is Nvidia RTX 2080Ti. The latency is averaged over 1000 inference steps and excludes the simulation time. We show that SECANT improves inference speed for both CARLA and DMControl by an order of magnitude in both environments.

5.5.5 iGibson: Indoor Object Navigation

iGibson [241, 195] is an interactive simulator with highly realistic 3D rooms and furniture (Fig. 5.1). The goal is to navigate to a lamp as closely as possible. The reward function incentivizes the agent to maximize the proportion of pixels that the lamp occupies in view, and success is achieved when this proportion exceeds 5% over 10 consecutive steps. Our agent is a virtual LoCoBot [67]. The action dimension is 2, which controls linear velocity and angular velocity. We use continuous action space with 10Hz control frequency.

Our benchmark features 1 training room and 20 test rooms, which include distinct furniture, layout, and interior design from training. The lamp is gray in training, but has much richer textures in testing. We construct 2 difficulty levels with 10 rooms each, depending on the extent of visual shift. The agent is randomly spawned in a room with *only* RGB observation (168×168), and outputs a 2D vector of linear and angular velocities.

We evaluate on each test room for 20 episodes and report success rates in Table 5.8. SAC without augmentation is better than SAC+crop because the lamp can be cropped out accidentally, which interferes with the reward function. Therefore we use plain SAC as the expert for SECANT. We consider this an edge case, since random cropping is otherwise broadly applicable. SECANT achieves **+15.8%** higher success rate than prior methods in unseen rooms.

5.6 Conclusion

Zero-shot generalization in visual RL has been a long-standing challenge. We introduce SECANT, a novel technique that addresses policy optimization and robust representation learning *sequentially*. We demonstrate that SECANT significantly outperforms prior SOTA in 4 challenging domains with realistic test-time variations. We also systematically study different augmentation recipes, strategies,

and distillation approaches. Compared to prior methods, we find that SECANT develops more robust visual representations and better task-specific saliency maps.

Chapter 6

Efficient Deployment of Visual Agents

6.1 Introduction

Analyzing videos to recognize human actions is a critical task for general-purpose video understanding algorithms. However, action recognition can be computationally costly, requiring processing of several frames spatiotemporally to ascertain the correct action. As embodied applications of video recognition for autonomous agents and mobile devices continue to scale, the need for efficient recognition architectures with fewer parameters and compute operations remains ever-growing.

Prior efforts for video action recognition [21, 221] rely on deep neural network architectures with expensive convolution operations across spatial and temporal context, which are often prohibitive for resource-constrained application domains. Recent work has proposed to improve the efficiency of the spatial and temporal operations separately [127, 169, 223, 246]. However, they are still largely bounded by the efficiency of the base spatial aggregation method through spatial convolutions.

In images, spatial shift operations [95, 238] have been proposed as a GPU-efficient alternative to traditional convolution operations, and architectures built using these operations have shown promise for efficient image recognition, though accuracy is limited. Recently, a temporal shift module (TSM) [127] based on hand-designed fixed temporal shift (Fig. 6.1, *top*) has been proposed to be used with existing 2D convolutional image recognition methods [232] for action recognition. However, the efficiency of their architecture family remains limited by the parameter and computation cost for spatial operations, as in other prior work.

A crucial observation is that much of the input spatiotemporal context contained in consecutive frames of a video sequence is often redundant to the action recognition task. Furthermore, the impact of modeling capacity on the action recognition task likely varies significantly at different

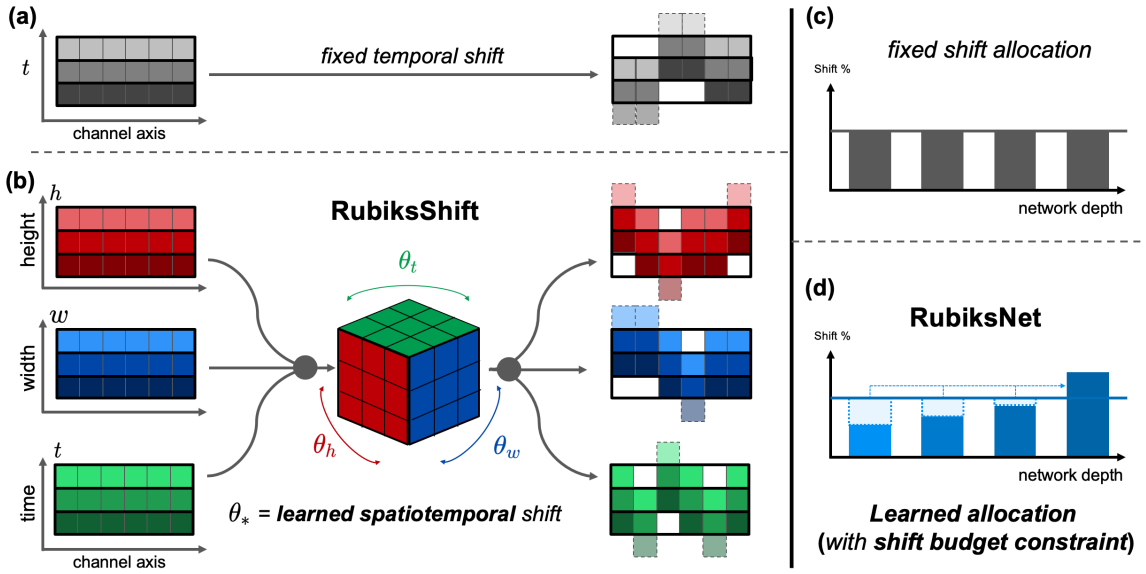


Figure 6.1: *Top:* Prior work for efficient shift-based video recognition [127] has explored **(a)** fixed temporal shift operation on expensive 2D convolution features, with **(c)** a fixed shift allocation network design. *Bottom:* We introduce **RubiksNet**, a new architecture based on a **(b)** learnable 3D-shift layer (RubiksShift) that learns to perform spatial (h, w) and temporal (t) shift operations jointly end-to-end, while also **(d)** learning an effective layer-wise network allocation of a constrained shift budget. Our model significantly improves the accuracy-efficiency boundary.

depths in the architecture. In other words, action recognition in videos poses a unique opportunity for pushing the boundaries of the efficiency-accuracy tradeoff curve by considering efficient shift operations in both space and time dimensions with flexible allocations.

However, a **key limitation** towards a naive generalization from fixed temporal shift [127] to a spatiotemporal shift scheme is that the design space becomes intractable. In particular, we would need to exhaustively explore the number of channels that are shifted for each dimension, the magnitude of these shifts, and the underlying layer design combining each of these operations, among other design choices. Thus, there is a clear motivation for enabling the architecture to learn the shift operations during training itself. Recent work [95] has explored the possibility of learning shifts for 2D image processing. However, the challenge of generalizing this formulation to enable stable and effective spatiotemporal optimization of shift-based operations on high-dimensional video input has remained unexplored.

To this end, we propose **RubiksNet**: a new video action recognition architecture based on a novel **spatiotemporal** shift layer (*RubiksShift*) that **learns** to perform shift operations jointly on spatial and temporal context. We explore several variations of our design, and find that our architecture **learns effective allocations** of shift operations within a **constrained shift budget**. We benchmark our overall approach on several standard action recognition benchmarks, including

large-scale temporal-focused datasets like Something-Something-v1 [63] and Something-Something-v2 [132], as well as others like UCF-101 [202] and HMDB [108]. We observe that our architecture can maintain competitive accuracy with the prior state-of-the-art on efficient shift-based action recognition [127] while simultaneously improving the efficiency and number of parameters by a large margin, up to 5.9x fewer parameters and 3.7x fewer FLOPs. Our controlled ablation analyses also demonstrate that these efficiency-accuracy gains come from the joint ability of our architecture to synergize spatial and temporal shift learning, as well as effective learned allocation of shift across the network.

6.2 Related Work

6.2.1 Action Recognition

The action recognition task in videos focuses on the classification of activities in video clips amongst a set of action classes. The initial set of deep network-based approaches processed frames individually using 2D convolutional neural networks [102, 199, 232]. For example, Temporal Segment Network (TSN) extracts features from sampled frames before averaging them for the final prediction [232]. While such methods are relatively efficient and parallelizable, they also do not model the temporal dynamics well. As such, the dominant paradigm in video action recognition is centered around the usage of spatial and temporal convolutions over the 3D space [21, 55, 96, 221]. There are also other action recognition works that exploit temporal information [251]. While these deep networks are more accurate, the increase in computational cost has proven to be substantial and prohibitive for efficiency-conscious applications.

Recent progress in action recognition has largely focused on two directions:

1. Improve efficiency by considering a mixture of 3D convolutions and separate spatial + temporal convolution operations [169, 223, 246, 262].

2. Incorporating longer term temporal reasoning [234, 262] to further improve the accuracy of the methods.

We differentiate this work from prior efforts along both axes. Along the first, we note that the above methods make progress towards bringing cubic scaling of 3D convolutions towards the quadratic scaling of 2D convolutions, but the spatial kernel remains an inherent strong bound on performance. Our approach offers an alternative that is much more efficient and has much fewer parameters by eliminating the need for 2D or 3D convolutions in the architecture. Along the second, our method introduces a learnable spatiotemporal shift operation that efficiently increases the effective receptive field capacity of the network, and allows it to flexibly allocate its capacity across the network, in contrast with prior efforts that leveraged fixed shift operations throughout the network [127].

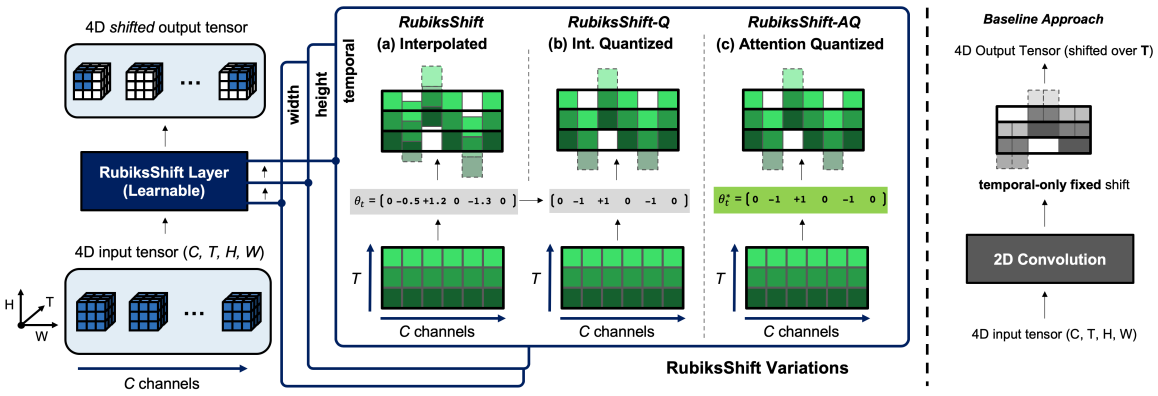


Figure 6.2: Our proposed RubiksShift layer (Section 6.3.1) is the primary driver of our overall RubiksNet architecture (Section 6.3.5). RubiksShift aims to perform a spatiotemporal shift operation on input 4D tensor along each of the input channels, for spatial and temporal axes. **(a)** Our primary RubiksShift layer is based on a continuous interpolated shift primitive, which enables a true gradient with low additional overhead, and **(b)** our RubiksShift-Q alternative is a quantized variant. **(c)** Finally, our RubiksShift-AQ variant enables learning integer shift using a temporal attention layer (also see Figure 6.3).

6.2.2 Efficient Neural Networks for Images and Video

Convolutions have been the main computational primitive in deep neural network approaches for computer vision tasks [33, 77, 86, 238, 260]. Recently, the shift operation has been proposed as a more hardware efficient alternative to spatial convolution for image classification [95, 238, 261]. While shift has been examined as a replacement for 1D temporal convolutions recently [127], the possibility of a learnable 3D shift operation has remained unexplored due to the challenges afforded by joint learning of the shift primitive across spatial and temporal context, and the traditional advantage spatial convolutions hold in action recognition architectures in terms of accuracy. We aim to propose a technique that need not depend on any spatial convolution operations during inference – only shift and pointwise convolution.

Additionally, our work remains complementary to literature on neural network compression [239], so while our proposed method saves substantially on model size and computation cost, it is possible that application of such techniques can lead to further gains. We also highlight work that ShuffleNet [260], MobileNet [86, 187], and SqueezeNet [88] for efficient 2D image classification. Tran et al. [222] factorizes 3D group convolutions while preserving channel interactions.

The aim of our work is to develop video models that can similarly be applied in embodied vision applications, where efficiency is essential.

6.3 Technical Approach

In this section, we describe our proposed method for efficient action recognition, where the task is to take an input video clip and classify which of a set of human action categories is present. We first describe our proposed RubiksShift layers for replacing spatiotemporal convolutions with learnable spatiotemporal shifts. Then, we detail how we compose these operations into the RubiksNet architecture.

6.3.1 RubiksShift: Learnable 3D Shift

Spatiotemporal Convolution

In a traditional convolutional network with 3D spatiotemporal convolutions, the input to each layer of the network is a 4D tensor $F \in \mathbb{R}^{C \times T \times H \times W}$, C is the number of input channels, T the temporal length, and H, W are the height and width respectively. The 3D spatiotemporal convolution operation [221] is then defined as:

$$O_{c',t,h,w} = \sum_{c,i,j,k} K_{c',c,k,i,j} F_{c,t+\hat{k},h+\hat{i},w+\hat{j}} \tag{6.1}$$

where $O \in \mathcal{R}^{C' \times T \times H \times W}$ is the output tensor, $K \in \mathbb{R}^{C' \times C \times T_K \times H_K \times W_K}$ is the 3D convolution kernel, i, j, k index along the temporal, height, and width dimensions of the kernel, and c, c' index along the channel dimensions. The indices $\hat{i}, \hat{j}, \hat{k}$ are the re-centered spatial and temporal indices, with $\hat{k} = k - \lfloor T_K/2 \rfloor, \hat{i} = i - \lfloor H_K/2 \rfloor, \hat{j} = j - \lfloor W_K/2 \rfloor$. Assuming $H = W$ for simplicity, the total number of parameters in this operation is thus $C \times C' \times T_K \times {H_K}^2$ and the computational cost is $C \times C' \times (T \times H^2) \times (T_K \times {H_K}^2)$. Indeed, this operation scales quadratically with the spatial input and cubically when considering temporal dimensions as well, both in parameters and number of operations.

Fixed Spatiotemporal Shift

In this context, we propose a spatiotemporal shift operation as an alternative to traditional 3D convolutions. Since the shift primitive proposed in prior work can be considered an efficient special case of depthwise-separable convolutions with fewer memory access calls [33, 238, 261], we can formalize the spatiotemporal shift operation as follows:

$$O'_{c,t,h,w} = \sum_{i,j,k} S_{c,k,i,j} F_{c,t+\hat{k},h+\hat{i},w+\hat{j}} \tag{6.2}$$

where $S \in \{0,1\}^{C \times T_K \times H_K \times W_K}$, such that $S_{c,k,i,j} = 1$ if and only if $(i, j, k) = (i_c, j_c, k_c)$, where i_c, j_c, k_c are the spatiotemporal shifts associated with each channel index c. Intuitively, if S is the

identity tensor, no shift occurs since every element maps back to itself. We note that in practice, this operation can be efficiently implemented by indexing into the appropriate memory address, meaning that the shift operation itself in this form requires no floating point operations (FLOPs). We then apply a pointwise convolution to the output of Eq. 6.2 to integrate the information across channels to obtain our final output:

$$O_{c',t,h,w} = \sum_{c} P_{c',c} O'_{c,t,h,w} \tag{6.3}$$

where P is the pointwise convolution kernel with only $C \times C'$ parameters. Assuming $H = W$, the computational cost is $C \times C' \times (T \times H^2)$, much lower than the spatiotemporal convolution. Notably, when deploying such an architecture, we can fuse the shift and pointwise convolution operations into a single efficient kernel call, meaning the final parameters and operation complexity from the spatiotemporal shift operation itself is subsumed entirely.

Learnable Spatiotemporal Shift

Finally, a key design aspect of our proposed spatiotemporal shift operation which differentiates itself from prior work in temporal modeling [127] is the ability of our model to *learn* the temporal shift primitive. In particular, our method learns to shift over the joint spatiotemporal context jointly, which affords the network the ability to efficiently consider a significant span of spatiotemporal context with fewer overall parameters.

Following prior work in spatial shift [95], we consider a continuous form of the traditionally discrete shift operation, allowing us to optimize the shift parameters directly by backpropagation. We define the 3D shift learnable parameters as:

$$\theta = \{(\gamma_c, \alpha_c, \beta_c) \mid c \in C\} \tag{6.4}$$

where $\gamma_c, \alpha_c, \beta_c$ are the temporal, vertical, and horizontal shift *parameters* for each channel c. With this, we consider an alternative formulation of Equation 6.2:

$$O'_{c,t,h,w} = \sum_{i,j,k} S^{\theta}_{c,k,i,j} F_{c,t+\hat{k},h+\hat{i},w+\hat{j}} \tag{6.5}$$

$$S^{\theta}_{c,k,i,j} = \prod_{(z,g)\in\{(\gamma_c,k),(\alpha_c,i),(\beta_c,j)\}} \begin{cases} \Delta z & \text{if } g = \lceil z \rceil, \\ 1 - \Delta z & \text{if } g = \lfloor z \rfloor, \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

with $\Delta z = z - \lfloor z \rfloor$ and g is the corresponding index to the z parameter dimension (e.g., γ_c corresponds to the time index k). Here, each sparse entry $S^{\theta}_{c,k,i,j}$ is a coefficient representing the

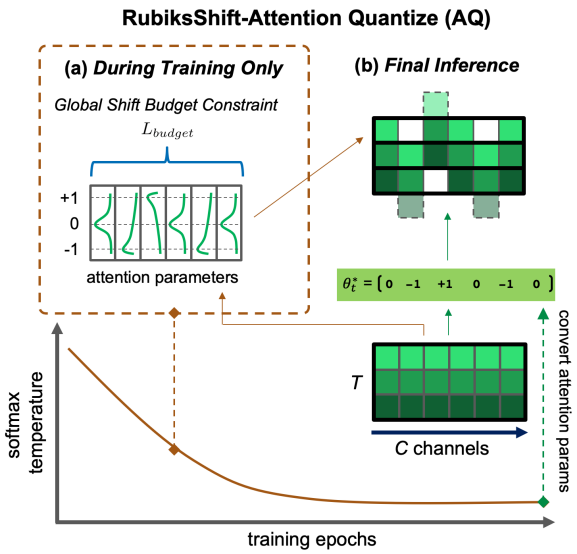


Figure 6.3: **(a)** During training, our RubiksShift-AQ variant parameterizes shift with an attention distribution corresponding to integer shift values. We can then train quantized temporal shift with a true gradient (under a budget constraint), in constrast with interpolated quantized methods [31]. Attention softmax temperature is annealed during training. **(b)** After training, the resulting one-hot attention parameters are converted to final integer shift values for efficient inference at test time.

product of interpolated shift contributions from all 3 dimensions. Intuitively, we are constructing a shift operation over an implicit continuous trilinearly-interpolated input activation tensor, where the interpolation is evaluated *sparsely* around the local neighborhood (2^3-point cube) around each shift operation at the location of each shift parameter. We note that this equation is once again a formalism; in practice, the operation can still be written efficiently on the GPU with minimal additional overhead with respect to discrete shift. We provide additional technical discussion in the supplement.

Taken together, we term this combined learnable spatiotemporal shift operation *RubiksShift* as shown in Figure 6.1. We observe that it enables our overall architecture to learn a joint spatiotemporal shift kernel that aggregates discriminative features over the full activation in an efficient manner.

6.3.2 Interpolated Quantized shift

While the interpolated formulation above remains efficient, we can push efficiency a little higher by considering a quantized version of the above spatiotemporal shift scheme. Our second RubiksShift variant is a *naive* spatiotemporal extension based on an interpolated shift quantization mechanism on images [31]. Briefly, during training, the model maintains a copy of the interpolated shift parameters as floats. In the forward pass, all shifts are rounded to the nearest integer. In the backward pass, gradient is computed with respect to the continuous shifts and these parameters are updated by regular gradient descent. Our ablative analysis shows that while this technique does work, the lack of a *true* gradient ends up hindering performance by a large margin.

6.3.3 Attention Quantized Shift

To address the shortcomings of the interpolated quantized variant, we propose RubiksShift-AQ (Fig. 6.3) as an alternative way to quantize temporal shifts with *exact* gradient. Related to concurrent attention pruning work [70] for 2D object recognition, we formulate *temporal attention shift* as an operator for video action recognition that parameterizes shift with an attention distribution corresponding to integer shift values. The attention weights are then annealed to one-hot, integer-valued shifts over the course of the training process. In this manner, the model is able to flexibly and stably learn a high-capacity spatiotemporal representation suitable for video action recognition, but at final inference time is as efficient as quantized shift. Given an attention weight tensor $\mathbf{W_{attn}}$, we compute:

$$\mathbf{W_{attnshift}} = softmax\left(\tau_a \frac{\mathbf{W_{attn}}}{std(\mathbf{W_{attn}})}\right) \tag{6.7}$$

where τ_a denotes the temperature of the softmax. After the softmax operation, every value in $\mathbf{W}$ is normalized between 0 and 1, which represents the attention weight. At high τ_a, the values in $\mathbf{W}$ are close to uniform, while at extremely low τ_a, $\mathbf{W}$ approaches a binary tensor with only one non-zero entry along each channel slice. During training, we anneal τ_a exponentially to a low value, typically 10^{-3}, at which stage we take the hard max operation to convert attention shift $\mathbf{W_{attnshift}}$ to the equivalent discrete integer shift operation S for efficient inference.

6.3.4 Shift Operations Budget Constraint

Prior work [127, 95] has noted that while shift operations save parameters and FLOPs, they can still incur latency cost through memory management. We incorporate this important aspect into our RubikShift design. A key feature of our proposed temporal attention shift is that we can apply a novel flexible constraint that corresponds to an overall "global shift budget", which penalizes the model for aggressively shifting. Importantly, the constraint is flexible in that it only penalizes a global metric for the shift – the specific allocation across layers at various depths in the network can then be learned by the network. Specifically, our budget constraint loss takes the form:

$$L_{budget} = \left|\left| \frac{1}{N_L} \sum_{l=1}^{N_L} \left(\frac{1}{C'} \sum_{c,*} \mathbf{W}^{(l)}_{\mathbf{nonzero}} \right) - B \right|\right| \tag{6.8}$$

where N_L is the number of layers, B is the shift budget between 0 and 1, and $\mathbf{W}^{(l)}_{\mathbf{nonzero}}$ denotes the attention weights in $\mathbf{W_{attnshift}}$ at layer l corresponding to the non-zero shift positions. We find that our proposed RubiksShift-AQ under budget constraint enables RubiksNet to discover interesting, non-uniform allocations of the shift budget across the network layers while preserving the global fraction of shift operations. To our knowledge, this is the first such method with learnable discrete shift under a shift resource constraint. Further, we find such learned allocation is critical to our

overall design in careful ablation analysis.

6.3.5 RubiksNet: Model Architecture Design

Our overall architecture mirrors the residual block design in the ResNet architecture [77]. Each RubiksShift layer includes shift operations as described in Sec 6.3.1 and pointwise Conv1×1 operations (Eq. 6.3) to facilitate information exchange across channels. We place RubiksShift layers at the "residual shift" position [127], which fuses temporal information inside a residual branch. While we choose ResNet-like structure as our backbone design, the RubiksShift operator is flexible to be plugged into any architecture to replace the heavy convolutional layers.

Stable Shift Training

To improve the stability of spatiotemporal shift training, we normalize the shift gradients to use only the direction of change [95], rather than the magnitude. In practice, equal normalization over all 3 axes is sub-optimal, because the spatial dimensions of video inputs (e.g., 224 × 224) are substantially larger than the temporal dimension (e.g., 8 frames). We propose *scaled* gradient normalization, which scales the gradient vector for the spatial shifts $\Delta\theta_h$, $\Delta\theta_w$ and temporal shift $\Delta\theta_t$ onto an ellipsoid rather than a unit sphere:

$$\overline{\Delta\theta_h} = \frac{\Delta\theta_h}{Z}; \quad \overline{\Delta\theta_w} = \frac{\Delta\theta_w}{Z}; \quad \overline{\Delta\theta_t} = \frac{\lambda\Delta\theta_t}{Z}, \tag{6.9}$$

where λ is the temporal gradient scaling factor and Z is the normalization factor:

$$Z = \sqrt{||\Delta\theta_h||^2 + ||\Delta\theta_w||^2 + \lambda||\Delta\theta_t||^2}. \tag{6.10}$$

Architecture Size Versions

We design several variants of RubiksNet models with different sizes to accommodate different computational budgets, analogous to prior work on shift for image classification [95]. Please refer to our supplementary material for full architecture breakdown tables. In our experiments (e.g., Table 6.1), we consider different size *classes* of our architecture, RubiksNet-Large, Medium, Small which all have the same channel width but different layer depths. These size classes are chosen to correspond with TSM [127] operating on standard ResNet-50, ResNet-34, and ResNet-18 backbones respectively. Our RubiksNet-Tiny model has the same depth as RubiksNet-Small, but a thinner width. We leverage this spectrum of models to generate our Pareto curves in Sec 5.5.

Table 6.1: Benchmark results on the Something-Something-v2 dataset [132]. RubiksNet offers strong performance across a range of base architectures as compared with TSM [127] architecture family. 2-clip accuracy metric per [127]; FLOPs reported for 1 clip, center crop for all architectures. (-) indicates value not reported. (#x) denotes efficiency savings factor relative to analogous size TSM model.

Method	Size	Input	1-Clip Val		2-Clip Val		#Param.	FLOPs/Video
			Top-1	Top-5	Top-1	Top-5		
TSN [232]	Large	8	30.0	60.5	30.4	61.0	24.3M	33G
TRN [262]	Large	8	48.8	77.6	-	-	18.3M	16G
bLVNet-TAM [53]	Large	8×2	59.1	86	-	-	25M	23.8G
TSM [127]	Large	8	58.8	85.6	61.3	87.3	24.3M	33G
	Medium	8	56.9	84.0	59.3	85.9	21.4M	29.4G
	Small	8	49.3	77.6	51.3	79.5	11.3M	14.6G
RubiksNet (Ours)	Large	8	59.0	85.2	61.7	87.3	8.5M (2.9x)	15.8G (2.1x)
	Medium	8	58.3	85.0	60.8	86.9	6.2M (3.5x)	11.2G (2.6x)
	Small	8	57.5	84.3	59.8	86.2	3.6M (3.1x)	6.8G (2.1x)
	Tiny	8	54.6	82.0	56.7	84.1	1.9M (5.9x)	3.9G (3.7x)

6.4 Experiments

In this section, we describe the experimental details (Sec. 6.4.1) and results of our method. In Sec 6.4.2, we detail our comparisons and analysis against the prior art methods on several standard benchmarks, and we show our architecture significantly pushes the state-of-the-art on the accuracy-efficiency frontier across large and smaller scale benchmarks. Finally, in Sec. 6.5, we conduct a series of controlled ablation studies and analysis to verify that the core scientific aspects of our architecture are responsible for this significant gain.

6.4.1 Experimental Setup

Overview

We leverage the Something-Something-v1 [63], Something-Something-v2 [132], UCF-101 [202], and HMDB [108] datasets to benchmark our approach. As a general rule, we follow training and evaluation protocols established in recent work [53, 127] for fair comparison, including input and metrics. We implement our RubiksNet architecture and training pipeline in PyTorch, and write the RubiksShift operators in CUDA and Pytorch C++ for efficiency.[1]

[1]See `rubiksnet.stanford.edu` project page for supplementary material.

Table 6.2: Benchmark results on the Something-Something-v1 dataset [63]. Results are reported as 1-clip accuracy; FLOPs are reported for 1 clip, center crop for all architectures. (-) indicates value not reported.

Method	Input	Val Top-1	Val Top-5	#Param.	FLOPs/Video
I3D [21]	64	45.8	76.5	12.7M	111G
NL I3D + GCN [235]	32+32	46.1	76.8	303M	62.2G
S3D [246]	64	47.3	78.1	8.8M	66G
bLVNet-TAM [53]	8×2	46.4	76.6	25M	23.8G
TSN [232]	8	19.5	-	10.7M	16G
TRN [262]	8	34.4	-	18.3M	16G
ECO [267]	8	39.6	-	47.5M	32G
TSM [127]	8	45.6	74.2	24.3M	33G
RubiksNet (Ours)	8	46.4	74.5	8.5M	15.8G

Spatial Shift Pretraining

Per prior works [53, 127], we pretrain the spatial portion of our RubiksNet models on ImageNet-1k [182] to reach comparable accuracy with spatial-shift image classification literature [95, 238]. Analogous to inflated convolution kernels [21], we initialize the spatial components of the 3D RubiksShift layers with the learned 2D shift patterns before benchmark training.

Something-Something-V2 and -V1

Something-Something-(v2,v1) are both large-scale datasets; SS-v2 in particular has 220k video clips and 174 action classes. The action labels, such as "pushing something from left to right" and "putting something next to something", cannot be predicted by looking at only a single frame. This challenging aspect separates this benchmark from similar large-scale benchmarks like Kinetics [103], as also noted in [127, 53]. Our spatial-pretrained RubiksNet is jointly trained end-to-end on the full benchmark, with the gradient normalization described in Eq. 6.9 for stability. For temporal attention shift, we initialize all the attention weights by sampling from $Uniform[1, 1.05]$, so that the initial attention distribution is roughly uniform over all possible shift locations. The softmax temperature is exponentially annealed from $T = 2.0$ to 10^{-3} over 40 epochs, before conversion to discrete integer shifts for evaluation.

UCF-101 and HMDB-51

These standard benchmarks have 101 and 51 action classes, respectively, and are smaller scale than the SS-v1/2 datasets. We follow the standard practice from prior work [127, 232] and pretrain our model on Kinetics [21] before fine-tuning on each benchmark. For fine-tuning, we follow the same general learning schedule protocol as Something-Something-v2, normalizing gradients again

Table 6.3: Quantitative results on UCF-101 [202] and HMDB-51 [108]. 2-clip metric, 1-clip FLOPs per [127]. Pareto curve results in Fig. 6.4 and supplement.

Method	Size	UCF-101		HMDB-51		#Param.	FLOPs
		Val Top-1	Val Top-5	Val Top-1	Val Top-5		
TSN [232]	Large	91.7	99.2	64.7	89.9	23.7M	33G
TSM [127]	Large	95.2	99.5	73.5	94.3	23.7M	33G
RubiksNet	Large	95.5	99.6	74.6	94.4	8.5M	15.8G

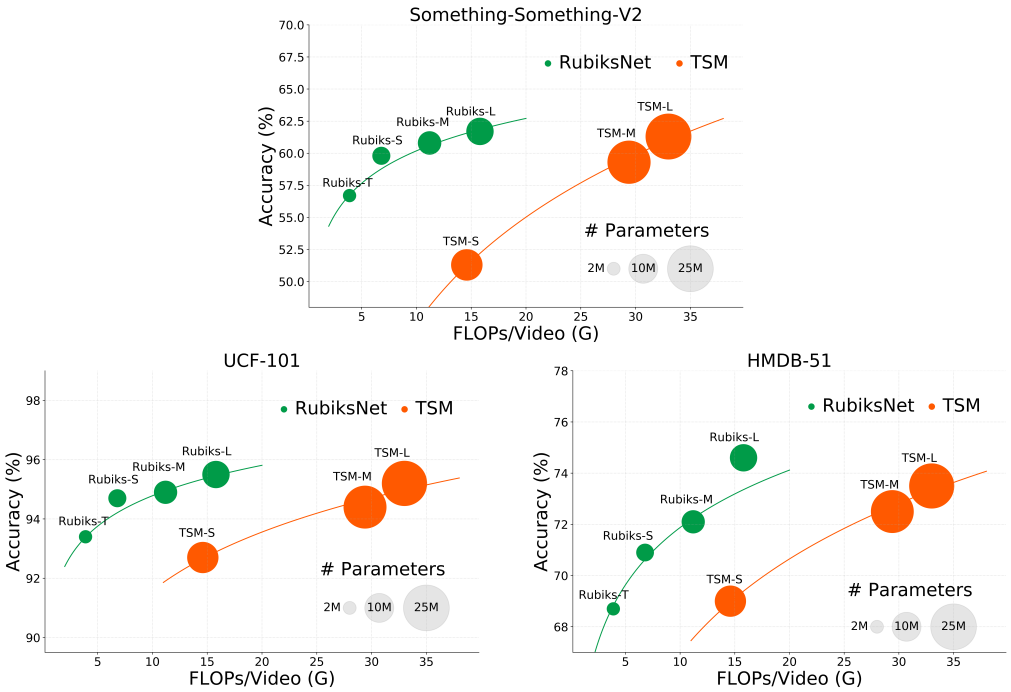


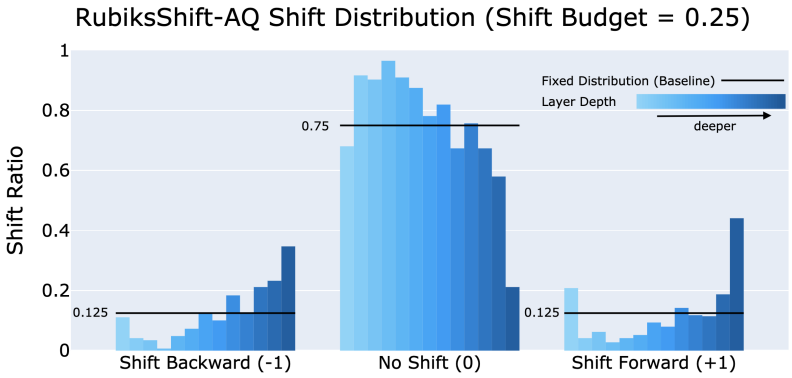
Figure 6.4: We report the Pareto curves for our method compared with prior work [127], with size of the circle corresponding to the number of model parameters, as per Tables 6.1-6.3. Our RubiksNet architecture family consistently offers better performance-efficiency tradeoff across datasets. (Additional vis. in supplement)

per Equation 6.9 and following a similar attention shift annealing schedule.

6.4.2 Benchmark Comparisons and Analysis

Baselines

Our key point of comparison is the recent state-of-the-art shift-based action recognition architecture TSM [127] from ICCV 2019. In contrast with our technique, TSM operates with a hand-designed,



Shift-T Budget (Latency)
(Ratio of Non-Zero Shift)

Layers	Rubiks-AQ	Fixed
all	**0.25**	**0.25**
1-4	0.23	0.25
5-12	0.15	0.25
13-48	0.23	0.25
49-51	0.74	0.25

Figure 6.5: We visualize our learned shifts distribution using our proposed attention quantized shift RubiksNet. The bottom labels are the different shifts $(-1, 0, +1)$ for a kernel size 3 RubiksShift-AQ temporal kernel, and the y-axis shows the proportion of channels with that temporal shift operation. Each colored bar represents a different layer, and we increase the depth of the network moving left to right. We observe that RubiksNet is able to learn to save its shift operations budget from early layers (few nonzero shift operations) to increase temporal modeling ability at deeper ones. Table 6.4 shows how this learned allocation consistently improves over heuristic techniques like TSM [127] that have a fixed shift budget allocation regardless of depth (shown by black horizontal bars above).

fixed shift approach on the time dimension only, with heuristics found by extensive architecture tuning. TSM also has a fixed allocation scheme across its network. In our benchmark comparisons, we also include comparisons against much heavier but well-known architectures, like I3D, S3D, and others [21, 235, 246, 53] for reference. Other networks, like TSN [232] and ECO [267] are also included as comparison points.

Evaluation

We follow the evaluation convention in prior work [127, 234, 235] and report results from two evaluation protocols. For "1-Clip Val" (Table 6.1), we sample only a single clip per video and the center 224×224 crop for evaluation. For "2-Clip Val", we sample 2 clips per video and take 3 equally spaced 224×224 crops from the full resolution image scaled to 256 pixels on the shorter side. 2-Clip evaluation yields higher accuracy, but requires more computation than 1-Clip evaluation. We employ the same protocol for all methods in all the tables.

Quantitative Analysis

In Tables 6.1-6.3, we demonstrate that our proposed architectures consistently achieve competitive or better accuracies than their baseline counterparts at a range of model capacities, while achieving significantly higher efficiency in both parameter counts and FLOPs. Additionally, we provide a detailed efficiency analysis breakdown in our supplement.

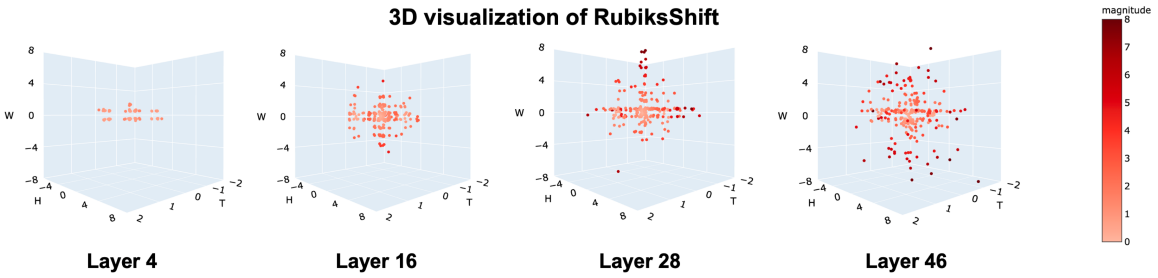


Figure 6.6: We visualize the overall learned interpolated shift distribution across spatial (H, W) and temporal (T) dimensions at different layers in the RubiksNet architecture. RubiksNet is conservative with shift operations in early layers, while increasing the 3D receptive field in deeper layers for better spatiotemporal modeling. Please refer to supplement for additional video visualizations.

We also benchmark several sizes of our model to draw a Pareto curve of performance and efficiency. In Figure 6.4, we visualize Pareto curves for our approach against TSM on multiple benchmarks. We observe a consistent trend that our method significantly improves the accuracy-efficiency tradeoff for efficient action recognition architectures.

On Something-Something-v2 datasets (Table 6.1), our most efficient model, RubiksNet-Tiny, outperforms TSM-Small by 5.3 absolute percentage points, while reducing parameters by 5.9x and FLOPs by 3.7x. This indicates that RubiksNet performs especially well in the low-resource regime when compared against prior work. Towards the other extreme, our highest-end model (RubiksNet-Large) consumes 24.1% *fewer* parameters and comparable FLOPs to the *lowest-end* baseline model (TSM-Small), while exceeding the latter's top 1 accuracy by more than 10 absolute percentage points.

Qualitative Analysis

In Figure 6.6, we visualize spatiotemporal shift operations across different layers of a trained RubiksNet architecture, showing how the model efficiently incorporates video context to provide an action recognition prediction by increasing its receptive field and shift operations deeper in the network. We include additional video visualizations and analysis in the supplement.

6.5 Ablations and Analysis

Finally, we provide controlled ablations and analysis over our core RubiksNet design principles. In particular, we verify that *jointly learning* our 3D-shift operations is key to our accuracy-efficiency improvements by synergizing both spatial and temporal shift learning. Futher, we verify our RubiksShift-AQ variant provides strong performance under a strict global shift (latency) budget.

Table 6.4: Ablation analysis: effect of learnable spatial and temporal shifts (Sec. 6.5). RubiksNet's ability to learn spatial and temporal shift jointly significantly improves over fixed, heuristic methods over spatial [238] and time [127] dimensions.

Spatial Shift Type	Temporal Shift Type	Val Top-1 (Large)	Val Top-1 (Small)
Learned	**Learned**	**71.4**	**69.5**
Learned	Fixed [127]	69.5	67.8
Fixed [238]	Learned	70.0	67.5
Fixed [238]	Fixed [127]	68.3	65.7

6.5.1 Ablation: Learned vs. Fixed

Our first ablation experiment provides a controlled analysis on the effect of learning the spatial and temporal shift aspects respectively. We describe our results in Table 6.4. We report top-1 accuracy results on the HMDB dataset for both Large and Small model size class. Critically, the architecture in all cases remains *constant* within a given model class size so there is no confounder due to different backbones. The only change is whether an aspect is "learned" or "fixed". In the fixed cases, we initialize the spatial and temporal shift parameters based on the heuristic initialization provided by ShiftNet [238] and TSM [127], respectively. Spatial and temporal learned cases are based on the RubiksNet (RubiksShift-AQ) method. For learned temporal, we set our budget constraint to 0.25 to exactly match that of the TSM [127] fixed setting for fair comparison and to ensure the same number of shift operations are performed. We find that our RubiksNet approach for learning spatial and temporal dimensions jointly consistently outperforms the ablations.

6.5.2 Ablation: Attention-Quantized RubiksShift

Our second ablation verifies the efficacy of the proposed RubiksShift layers and its variations. We report our results in Table 6.5. Here, we highlight that our RubiksShift-AQ is able to achieve comparable accuracy with a budget constraint of only 0.125 shift ratio, in comparison to the full RubikShift variant. In contrast with the naive RubiksShift-IQ variant, RubiksShift-AQ enables discrete shift learned with true gradient and substantially outperforms.

We observe that given a shift budget constraint, our attention shift mechanism is able to learn nontrivial temporal patterns (Figure 6.5) without hand-engineered prior knowledge. The network chooses to allocate more non-zero shifts for deeper layers, likely because heavier information exchange in the more abstract feature space is beneficial to the network's temporal modeling capability. Such findings are in alignment with traditional hand-designed "top-heavy" spatiotemporal convolutional networks [246].

Importantly, RubiksNet's learned temporal shift pattern can be thought of as an allocation of the limited "temporal modeling power budget". Prior works like [246] enumerate many configurations

Table 6.5: Ablation analysis: impact of RubiksShift design (Sec. 6.5). Our attention-quantized variant is able to learn discrete shift operations with comparable performance to full interpolated, while observing shift/latency budget constraints.

RubiksShift Type	Val Top-1	Exact Gradient	Integer Shift	Budget
Interpolated (RS)	61.7	✓		
Interpolated Quantized (RS-IQ)	58.2		✓	
Attention Quantized (RS-AQ)	61.6	✓	✓	✓ (0.125)

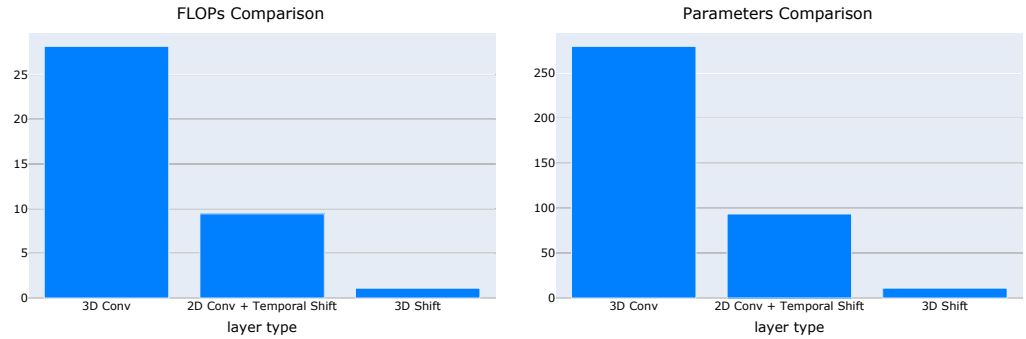


Figure 6.7: Efficiency comparison at layer level. Our RubiksShift (3D learnable shift) layer shows a large efficiency gain over analogous 3D convolution and Shift+2D convolution prior work. See Sec. 6.5.3.

of temporal modeling allocation (i.e. permutations of Conv2D and Conv3D layers), and test them individually to find the best candidate. In contrast, our proposed method discovers a good temporal allocation pattern from random temporal initialization.

6.5.3 Efficiency Analysis

In this section, we provide additional details and analysis of efficiency of our models, breaking down the contribution of our 3D RubikShift block from an operations perspective with respect to traditional 3D Convolution as well as the recent 2D Convolution + Shift (TSM) block from ICCV 2019. We also provide **runtime analysis** of our method.

FLOPs and Parameters Protocol

Our FLOP and parameter computation procedure aligns with prior work [127] for consistency. In a RubiksShift layer, the main contributor to the FLOP count is the 1x1 pointwise convolution layers, which are dramatically less expensive than traditional 2D or 3D conv. The traditional shift operation itself is considered zero-FLOP, since it can be fused into the pointwise convolution as one GPU kernel call [238]. Learnable shift incurs small FLOP/param cost, but otherwise similarly

Table 6.6: Runtime Latency Comparison; Block types correspond with Figure 6.7. See Section 6.5.3 for details.

Block Type	Runtime Latency
3D Conv	7.98ms ± 0.75ms
2D Conv + Shift (TSM)	3.59ms ± 0.13ms
3D Shift (Ours)	**0.90ms ± 0.12ms**

efficient when properly implemented.

Visualization of Efficiency Breakdown

We visualize a full efficiency analysis breakdown of our RubiksShift layer in Figure 6.7 and 6.8. For our analysis here, we **control** the **same input** ($(T, H, W) = (8, 112, 112)$) and **input/output channels** (input and output is fixed to 72 channels for all blocks, so that channel count does not affect the analysis) for all calculations. Further, all blocks here are standard blocks with consistent channels throughout the block. Figure 6.7 shows the total cost comparison; we calculate that a RubiksShift layer has ∼25x fewer FLOPs/params in contrast with traditional 3D, and ∼8x fewer than TSM (shift + 2D conv). Figure 6.8 shows the breakdown by percentage of FLOPs. These gains translate to our overall **RubiksNet** architecture, our gains are chiefly due to the replacement of *all* spatial and spatiotemporal convolutions with a learnable shift-based operation. We note that the full *RubiksNet* numbers described in the main paper (which are relatively lower, but show significant improvement) also account for all the extra layers in the full architecture (e.g. the fully-connected layers, which are not replaced by RubiksShift blocks).

Runtime/Latency

We also report latency analysis in the Table 6.6. We benchmark each layer/block type for runtime on the same GPU and hardware set-up (single GPU, Titan Xp) and averaged over 100 trials. Our input tensor in all cases is $(N, T, C, H, W) = (8, 8, 72, 56, 56)$ (batch size N is 8), and architecture blocks are similarly controlled for same input/output channels as in our other efficiency analysis breakdown. We observe that our 3D RubiksShift method has consistently better runtime than prior work.

6.6 Conclusion

We introduced RubiksNet, a new efficient architecture for video action recognition. We examined the potential for a model based on our proposed 3D-spatiotemporal RubiksShift operations, and explored several novel variations of our design that enable stable joint training with flexible shift budget allocation. We benchmarked our method on several standard action recognition benchmarks, and

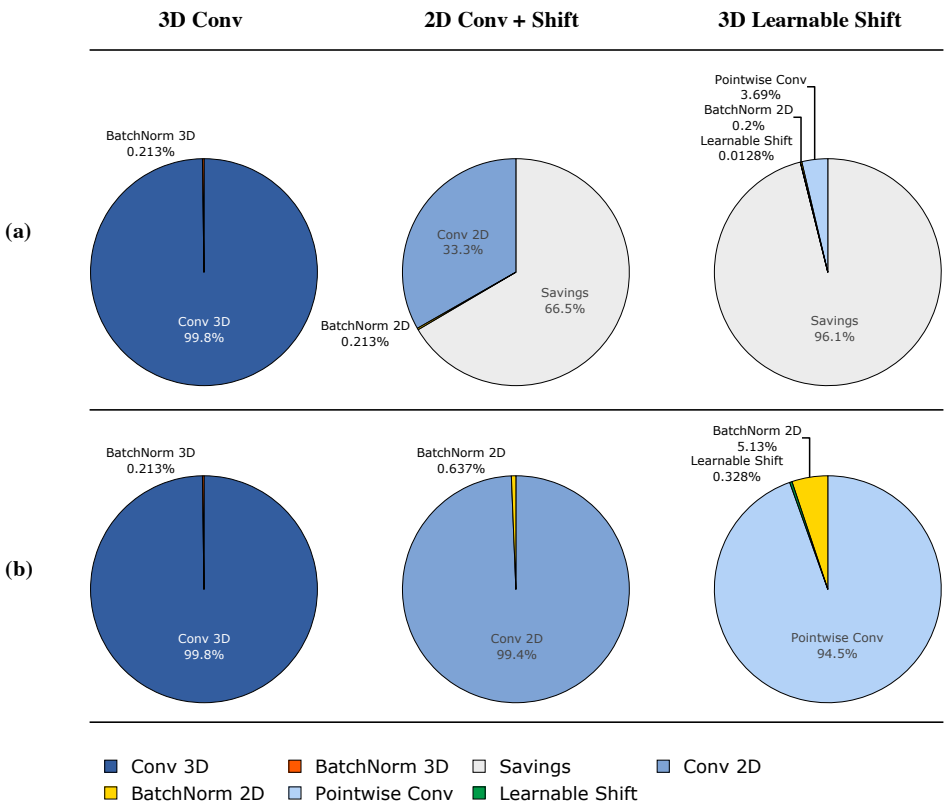


Figure 6.8: Breakdown of FLOPs by percentage for RubiksShift (Learnable 3D Shift) against 3D Conv and 2D Conv+Shift [127] analogous blocks, controlling for channel/input. **(a)** shows breakdown of FLOPs for all three blocks normalized to the 3D conv block. The "savings" section indicates the saved relative compute. **(b)** is normalized to itself. See Sec. 6.5.3.

find that RubiksNet can match or exceed the accuracies given by the previous state-of-the-art shift-based action recognition architecture at a fraction of the parameter and FLOP cost. Through careful and controlled ablations, we verified these gains are rooted in our core architecture contributions, from the joint learning of the spatial and temporal shifts to RubiksNet's ability to learn a flexible allocation of shift budget to maximize accuracy at minimal shift cost.

Chapter 7

Conclusion

7.1 Summary

In this dissertation, we have explored an effective recipe towards developing algorithms and systems that are able to train and deploy visual agents at scale: train the agents in rich simulation, then overcome the sim-to-real gap, and finally deploy efficiently on resource-constrained devices with lightweight video processing architectures.

To recapitulate, we start with SURREAL, an open-source distributed framework that provides a full-stack solution to accelerate reinforcement learning (RL) significantly for complex robotics tasks. We take a deep dive into the systems design of SURREAL that enables the superb performance and scalability. SURREAL is capable of deploying to a wide range of hardware from a personal laptop to full-fledged cloud clusters. The learning performances of our distributed algorithms establish new state of the art in visual policy learning. Next, we construct iGibson, an ecologically valid and visually realistic simulator for home robotic tasks. We then introduce SECANT, a novel policy learning method that achieves zero-shot generalization to unseen visual environments with large distributional shifts, which facilitates sim-to-real transfer. Finally, we design RubiksNet, a new family of video learning architectures that unlocks deep video understanding for visual agents on edge devices.

7.2 Future Directions

Looking forward, there are many new promising techniques to scale up visual agent learning even further. I hope to explore these exciting ideas in the next stage of my career:

- **Scaling up the simulation and tasks**: although the community has made tremendous progress in more realistic and expansive simulators [195, 46, 189], we are still far from photo-realistic rendering and highly complex tasks that resemble those in the real world. In order to provide higher quality data to our neural network agents, we need much more engineering efforts on improving rendering efficiency and task richness. I look forward to the fruition of several ongoing initiatives, such as NVIDIA Omniverse [152], Habitat 2.0 suite [212], and BEHAVIOR large-scale household tasks [205].
- **Scaling up the architecture**: the AI community has witnessed the unreasonable effectiveness of gigantic sequence learning models, such as GPT-3 [16] that performs natural language understanding in a few-shot manner, CLIP [171] that associates language caption and images, and DALL-E [176] that generates novel images conditioned on a textual description. Visual robotics, however, has not experienced similar quantum leaps in modeling capabilities. I believe high-capacity models like the Transformer [226] will help the embodied agents understand the visual and physical world far better than the current state of the art, and enable strong compositional generalization to novel scenarios.
- **Scaling up the learning objective**: a large and flexible neural network policy will not be able to fully utilize its modeling capacity without an equally scalable learning objective. So far, many reinforcement learning works [194, 49] still require manually engineered reward functions to learn meaningful behavior. I see self-supervised learning [44] as a powerful new paradigm to scale up visual agent learning far beyond the current limits. Some promising candidates are autoregressive sequence objective [28, 16] and contrastive learning [75, 30, 65, 20].

7.3 Closing Thoughts

Ever since my childhood, AI has never failed to fascinate me. The idea that deep thoughts could emerge from the symphony of 1's and 0's always takes my breath away. Throughout my quest at Stanford, I have come to appreciate that AI possesses a unique combination of scientific inquiry and raw creativity. Unlike math and physics, which typically have a clear distinction between right and wrong, in AI there is no "correct" way. Of course we can attempt to mimic the brain, but we do not have to, much like airplanes do not need to flap their wings like birds. In AI research, we are encouraged to be creative, to look at places where no one else has looked before, and sometimes are even allowed to distort the reality a little to discover the key.

Yet this kind of creativity does not come from nowhere. With years of rigorous and painstaking training, reading literally thousands of papers, and digesting over half a century of prior works – only then am I able to build the castle of imagination on top of the shoulder of giants.

My Ph.D. journey has come to an end, but in the grand scheme of life, it is but a milestone. I believe in the ultimate goal that AI will lead all humanity to a better world. I aspire to be among the pioneers who make this a reality. It is a dream so intimately known, profoundly felt, deeply loved, and absolutely committed to.

My real journey is just getting started.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[3] Peter Anderson et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[4] 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.

[5] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[6] Himani Arora, Rajath Kumar, Jason Krone, and Chong Li. Multi-task learning for continuous control. *arXiv preprint arXiv:1802.01034*, 2018.

[7] Christopher G Atkeson, BPW Babu, N Banerjee, D Berenson, CP Bove, X Cui, M DeDonato, R Du, S Feng, P Franklin, et al. What happened at the darpa robotics challenge, and why. *arXiv preprint*, 2016.

[8] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, pages 2930–2941, 2018.

[9] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning thorugh asynchronous advantage actor-critic on a GPU. In *ICLR*, 2017.

[10] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1615–1620. IEEE, 2001.

[11] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *ArXiv*, abs/1612.03801, 2016.

[12] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019.

[13] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019.

[14] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 521–528. IEEE, 2000.

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

[16] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[17] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Queue*, 14(1):10, 2016.

[18] Caelan Reed Garrett. PyBullet Planning. `https://pypi.org/project/pybullet-planning/`, 2018.

[19] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, et al. The YCB object and model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics*, 2015.

[20] Mathilde Caron, Hugo Touvron, Ishan Misra, Herv'e J'egou, J. Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *ArXiv*, abs/2104.14294, 2021.

[21] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[22] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8001–8008, 2019.

[23] Angel Chang et al. Matterport3d: Learning from rgb-d data in indoor environments. In *2017 International Conference on 3D Vision (3DV)*, pages 667–676. IEEE, 2017.

[24] Angel X Chang et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[25] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *ICRA*, 2017.

[26] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[27] Kevin Chen, Juan Pablo de Vicente, Gabriel Sepulveda, Fei Xia, Alvaro Soto, Marynel Vázquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. *arXiv preprint arXiv:1903.00445*, 2019.

[28] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

[29] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: End-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799*, 2018.

[30] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *ArXiv*, abs/2002.05709, 2020.

[31] Weijie Chen, Di Xie, Yuan Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7241–7250, 2019.

[32] Changhyun Choi and Henrik I Christensen. Rgb-d object pose estimation in unstructured environments. *Robotics and Autonomous Systems*, 75:595–613, 2016.

[33] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[34] Alfredo V Clemente, Humberto N Castejón, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

[35] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

[36] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

[37] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, et al. Analysis and observations from the first amazon picking challenge. *IEEE Trans. on Automation Science and Engineering*, 2016.

[38] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[39] Wojciech Marian Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant M Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. *arXiv preprint arXiv:1902.02186*, 2019.

[40] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[41] Angel P del Pobil, Rad Madhavan, and Elena Messina. Benchmarks in robotics research. In *Workshop IROS*. Citeseer, 2006.

[42] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[43] Developers. Nadrin/pbr. `https://github.com/Nadrin/PBR`. Accessed: 2020-10-30.

[44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[45] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[46] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[47] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *ICML*, 2016.

[48] Zackory Erickson, Vamsee Gangaram, Ariel Kapusta, C Karen Liu, and Charles C Kemp. Assistive gym: A physics simulation framework for assistive robotics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10169–10176. IEEE, 2020.

[49] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[50] Linxi Fan, Shyamal Buch, Guanzhi Wang, Ryan Cao, Yuke Zhu, Juan Carlos Niebles, and Li Fei-Fei. Rubiksnet: Learnable 3d-shift for efficient video action recognition. In *European Conference on Computer Vision*, pages 505–521. Springer, 2020.

[51] Linxi Fan, Guanzhi Wang, De-An Huang, Zhiding Yu, Li Fei-Fei, Yuke Zhu, and Animashree Anandkumar. Secant: Self-expert cloning for zero-shot generalization of visual policies. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3088–3099. PMLR, 18–24 Jul 2021.

[52] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal-system: Fully-integrated stack for distributed deep reinforcement learning. *arXiv preprint arXiv:1909.12989*, 2019.

[53] Quanfu Fan, Chun-Fu Richard Chen, Hilde Kuehne, Marco Pistoia, and David Cox. More is less: Learning efficient video representations by big-little network and depthwise temporal aggregation. In *Advances in Neural Information Processing Systems*, pages 2261–2270, 2019.

[54] Jesse Farebrother, Marlos C. Machado, and Michael H. Bowling. Generalization and regularization in dqn. *ArXiv*, abs/1810.00123, 2018.

[55] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *Advances in neural information processing systems*, pages 3468–3476, 2016.

[56] Norman Ferns and Doina Precup. Bisimulation metrics are optimal value functions. In *UAI*, pages 210–219. Citeseer, 2014.

[57] Huan Fu, Bowen Cai, Lin Gao, Lingxiao Zhang, Cao Li, Zengqi Xun, Chengyue Sun, Yiyun Fei, Yu Zheng, Ying Li, et al. 3d-front: 3d furnished rooms with layouts and semantics. *arXiv preprint arXiv:2011.09127*, 2020.

[58] Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. *ArXiv*, abs/1806.07377, 2019.

[59] 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.

[60] Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. Vrkitchen: an interactive 3d virtual environment for task-oriented learning. *arXiv preprint arXiv:1903.05757*, 2019.

[61] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.

[62] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[63] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax, and Roland Memisevic. The "something something" video database for learning and evaluating visual common sense. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[64] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. In *International Conference on Machine Learning*, pages 1792–1801. PMLR, 2018.

[65] Jean-Bastien Grill, Florian Strub, Florent Altch'e, C. Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, B. A. Pires, Z. Guo, M. G. Azar, Bilal Piot, K. Kavukcuoglu, R. Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *ArXiv*, abs/2006.07733, 2020.

[66] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 2016.

[67] Abhinav Gupta, Adithyavairavan Murali, Dhiraj Prakashchand Gandhi, and Lerrel Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. *Advances in Neural Information Processing Systems*, 31:9094–9104, 2018.

[68] Tuomas Haarnoja et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[69] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018.

[70] Ghouthi Boukli Hacene, Carlos Lassance, Vincent Gripon, Matthieu Courbariaux, and Yoshua Bengio. Attention Based Pruning for Shift Networks. *arXiv:1905.12300 [cs]*, May 2019.

[71] Danijar Hafner, James Davidson, and Vincent Vanhoucke. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv preprint arXiv:1709.02878*, 2017.

[72] Nicklas Hansen, Yu Sun, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[73] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. *arXiv preprint arXiv:2011.13389*, 2020.

[74] Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE international conference on robotics and automation*, pages 2493–2498. IEEE, 2010.

[75] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B Girshick. Momentum contrast for unsupervised visual representation learning. corr abs/1911.05722 (2019). *arXiv preprint arxiv:1911.05722*, 2019.

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

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

[78] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[79] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[80] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and surface variations. *arXiv: Learning*, 2018.

[81] Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *International Conference on Learning Representations*, 2019.

[82] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[83] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics and Automation Letters*, 4(4):3184–3191, 2019.

[84] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

[85] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[86] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[87] Yinlin Hu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. Segmentation-driven 6d object pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3385–3394, 2019.

[88] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[89] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *Advances in Neural Information Processing Systems*, 32:13978–13990, 2019.

[90] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Böhmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *ArXiv*, abs/2006.05826, 2020.

[91] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[92] Stephen James, Andrew J. Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv preprint arXiv:1707.02267*, 2017.

[93] 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.

[94] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[95] Yunho Jeon and Junmo Kim. Constructing fast network through deconstruction of convolution. In *Advances in Neural Information Processing Systems*, pages 5951–5961, 2018.

[96] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[97] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, et al. Caffe: Convolutional architecture for fast feature embedding. In *22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[98] Minqi Jiang, Ed Grefenstette, and Tim Rocktäschel. Prioritized level replay. *arXiv preprint arXiv:2010.03934*, 2020.

[99] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv: Learning*, 2018.

[100] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Scandinavian Conference on Image Analysis*, pages 28–40. Springer, 2019.

[101] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. *International Conference on Robotics and Automation (ICRA)*, 2019.

[102] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[103] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[104] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[105] Eric Kolve et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[106] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[107] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[108] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2556–2563. IEEE, 2011.

[109] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[110] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Department of Computer Science; Iowa State University, 1998.

[111] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

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

[113] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[114] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500, 2018.

[115] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

[116] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020.

[117] Youngwoon Lee, Edward S Hu, Zhengyu Yang, Alex Yin, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019.

[118] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O Stanley. Es is more than just a traditional finite-difference approximator. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 450–457. ACM, 2018.

[119] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

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

[121] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016.

[122] Eric Li, Roberto Martín-Martín, Fei Xia, and Silvio Savarese. Hrl4in: Hierarchical reinforcement learning forinteractive navigation with mobile manipulators. In *2019 Conference on Robot Learning (CoRL)*, 2019.

[123] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. arxiv e-prints, page. *arXiv preprint arXiv:1712.09381*, 2017.

[124] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, 2017.

[125] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. *arXiv preprint arXiv:1810.05762*, 2018.

[126] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *ICLR*, abs/1509.02971, 2016.

[127] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding, 2018.

[128] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[129] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[130] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael H. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. In *IJCAI*, 2018.

[131] Raj Madhavan, Rolf Lakaemper, and Tamás Kalmár-Nagy. Benchmarking and standardization of intelligent robotic systems. In *International Conference on Advanced Robotics*, pages 1–7, 2009.

[132] Farzaneh Mahdisoltani, Guillaume Berger, Waseem Gharbieh, David Fleet, and Roland Memisevic. On the effectiveness of task granularity for transfer learning. *arXiv preprint arXiv:1804.09235*, 2018.

[133] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

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

[135] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural autonomous navigation with riemannian motion policy. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8860–8866. IEEE, 2019.

[136] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Scaling local control to large-scale topological navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 672–678. IEEE, 2020.

[137] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[138] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[139] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *ICML*, 2014.

[140] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[141] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

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

[143] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019.

[144] Moley. Moley robotic chef, 2021.

[145] Thierry Moreau, Tianqi Chen, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Vta: An open hardware-software stack for deep learning. *arXiv preprint arXiv:1807.04188*, 2018.

[146] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. *arXiv preprint*, 2017.

[147] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[148] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[149] Iman Nematollahi, Oier Mees, Lukas Hermann, and Wolfram Burgard. Hindsight for foresight: Unsupervised structured dynamics models from physical interaction. *arXiv preprint arXiv:2008.00456*, 2020.

[150] NVIDIA. Nvidia jetson platform, 2021.

[151] NVIDIA. Nvidia jetson platform, 2021.

[152] NVIDIA. Nvidia omniverse, 2021.

[153] ODE team. Ode. `https://ode.org/`. Accessed: 2020-10-30.

[154] M Andrychowicz OpenAI, 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. *arXiv preprint arXiv:1808.00177*, 2(3):5–1, 2018.

[155] Organizers. igibson sim2real challenge, cvpr2020.

[156] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Xiaodong Song. Assessing generalization in deep reinforcement learning. *ArXiv*, abs/1810.12282, 2018.

[157] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[158] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *arXiv preprint*, 2017.

[159] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[160] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[161] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *arXiv preprint arXiv:1710.06537*, October 2017.

[162] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[163] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *AAAI*. Atlanta, 2010.

[164] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IROS*, pages 2219–2225. IEEE, 2006.

[165] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. *ArXiv e-prints*, 2017.

[166] Ivaylo Popov, Nicolas Heess, Timothy P. Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, et al. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[167] Lorenzo Porzi, Samuel Rota Bulo, Adrian Penate-Sanchez, Elisa Ricci, and Francesc Moreno-Noguer. Learning depth-aware deep representations for robotic perception. *IEEE Robotics and Automation Letters*, 2(2):468–475, 2016.

[168] X Puig, K Ra, M Boben, J Li, T Wang, S Fidler, and A Torralba. Virtualhome: Simulating household activities via programs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8494–8502, 2018.

[169] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *proceedings of the IEEE International Conference on Computer Vision*, pages 5533–5541, 2017.

[170] Ahmed Hussain Qureshi and Yasar Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.

[171] 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. *arXiv preprint arXiv:2103.00020*, 2021.

[172] Antonin Raffin. Rl baselines zoo. `https://github.com/araffin/rl-baselines-zoo`, 2018.

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

[174] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *ArXiv*, abs/2002.12292, 2020.

[175] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham M. Kakade. Towards generalization and simplicity in continuous control. *ArXiv*, abs/1703.02660, 2017.

[176] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[177] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11157–11166, 2020.

[178] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[179] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3674–3683, 2020.

[180] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

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

[182] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[183] Andrei Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

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

[185] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. *RSS*, 2017.

[186] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[187] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[188] Manolis Savva, Angel X Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017.

[189] 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.

[190] Stefan Schaal et al. Learning from demonstration. *Advances in neural information processing systems*, pages 1040–1046, 1997.

[191] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246. Wiley Online Library, 1994.

[192] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[193] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 2016.

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

[195] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martın-Martın, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P Tchapmi, Kent Vainio, Li Fei-Fei, and Silvio Savarese. igibson 1.0, a simulation environment for interactive tasks in large realistic scenes. *arXiv preprint*, 2020.

[196] William B Shen, Danfei Xu, Yuke Zhu, Leonidas J Guibas, Li Fei-Fei, and Silvio Savarese. Situational fusion of visual representation for visual navigation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2881–2890, 2019.

[197] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[198] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[199] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[200] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020.

[201] Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. *ArXiv*, abs/1912.02975, 2020.

[202] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[203] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, abs/2004.04136, 2020.

[204] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[205] 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.

[206] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

[207] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[208] Surreal. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Proceedings of the 2018 Conference on Robot Learning (CoRL)*, 2018.

[209] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[210] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, pages 216–224, 1990.

[211] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[212] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021.

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

[214] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. Technical report, DeepMind, January 2018.

[215] Unity team. Unity. `https://unity.com/`. Accessed: 2020-10-30.

[216] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.

[217] Tesla. Tesla autopilot, 2021.

[218] 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. *arXiv preprint arXiv:1703.06907*, pages 23–30, 2017.

[219] Joshua Tobin, Rachel H Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, Sep 2017.

[220] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[221] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[222] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. Video classification with channel-separated convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5552–5561, 2019.

[223] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

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

[225] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

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

[227] Oriol Vinyals, I. Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, J. Chung, David H. Choi, Richard Powell, Timo Ewalds, P. Georgiev, Junhyuk Oh, Dan Horgan, M. Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, J. Agapiou, Max Jaderberg, A. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, D. Budden, Yury Sulsky, James Molloy, T. Paine, Caglar Gulcehre, Ziyu Wang, T. Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[228] Chen Wang, Roberto Martín-Martín, Danfei Xu, Jun Lv, Cewu Lu, Li Fei-Fei, Silvio Savarese, and Yuke Zhu. 6-pack: Category-level 6d pose tracker with anchor-based keypoints. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10059–10066. IEEE, 2020.

[229] Huan Wang, Stephan Zheng, Caiming Xiong, and Richard Socher. On the generalization gap in reparameterizable reinforcement learning. In *ICML*, 2019.

[230] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[231] Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *arXiv preprint arXiv:2010.10814*, 2020.

[232] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision*, pages 20–36. Springer, 2016.

[233] Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. Shape2motion: Joint analysis of motion parts and attributes from 3d shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8876–8884, 2019.

[234] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[235] Xiaolong Wang and Abhinav Gupta. Videos as space-time region graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 399–417, 2018.

[236] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[237] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3381–3387. IEEE, 2008.

[238] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9127–9135, 2018.

[239] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6026–6035, 2018.

[240] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. *arXiv preprint arXiv:2008.07792*, 2020.

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

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

[243] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[244] 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.

[245] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on robot learning*, pages 1369–1378. PMLR, 2020.

[246] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 305–321, 2018.

[247] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[248] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *arXiv preprint arXiv:1610.00673*, 2016.

[249] Mengyuan Yan, Qingyun Sun, Iuri Frosio, Stephen Tyree, and Jan Kautz. How to close sim-real gap? transfer with segmentation! *arXiv preprint arXiv:2005.07695*, 2020.

[250] Jiachen Yang, Brenden Petersen, Hongyuan Zha, and Daniel Faissol. Single episode policy transfer in reinforcement learning, 2019.

[251] Guangle Yao, Tao Lei, and Jiandan Zhong. A review of convolutional-neural-network-based action recognition. *Pattern Recognition Letters*, 118:14–22, 2019.

[252] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *ArXiv*, abs/1910.01741, 2019.

[253] 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, 2020.

[254] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6023–6032, 2019.

[255] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[256] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *ArXiv*, abs/1806.07937, 2018.

[257] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[258] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[259] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[260] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[261] Huasong Zhong, Xianggen Liu, Yihui He, Yuchun Ma, and Kris Kitani. Shift-based primitives for efficient convolutional neural networks. *arXiv preprint arXiv:1809.08458*, 2018.

[262] Bolei Zhou, Alex Andonian, Aude Oliva, and Antonio Torralba. Temporal relational reasoning in videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 803–818, 2018.

[263] Brady Zhou, Nimit Kalra, and Philipp Krähenbühl. Domain adaptation through task distillation. In *European Conference on Computer Vision*, pages 664–680. Springer, 2020.

[264] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, et al. Target-driven visual navigation in indoor scenes using deep rl. In *International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[265] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *RSS*, 2018.

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

[267] Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. Eco: Efficient convolutional network for online video understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 695–712, 2018.