Title

Scalable Lifelong Imitation Learning for Robot Fleets

Author

Hoque, Ryan

Publication Date

2024

Peer reviewed|Thesis/dissertation

Scalable Lifelong Imitation Learning for Robot Fleets

by

Ryan Hoque

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Ken Goldberg, Chair
Pieter Abbeel
Anca Dragan
Shuran Song

Spring 2024

Scalable Lifelong Imitation Learning for Robot Fleets

Abstract

Scalable Lifelong Imitation Learning for Robot Fleets

by

Ryan Hoque

Doctor of Philosophy in Computer Science

University of California, Berkeley

Ken Goldberg, Chair

Recent breakthroughs in deep learning have revolutionized natural language processing, computer vision, and robotics. Nevertheless, reliable robot autonomy in unstructured environments remains elusive. Without the Internet-scale data available for language and vision, robotics faces a unique chicken-and-egg problem: robot learning requires large datasets from deployment at scale, but robot learning is not yet reliable enough for deployment at scale. We propose a scalable human-in-the-loop learning paradigm as a potential solution to this paradox, and we argue that it is the key ingredient behind the recent growth of large-scale robot deployments in applications such as autonomous driving and e-commerce order fulfillment. We develop novel formalisms, algorithms, benchmarks, systems, and applications for this setting and evaluate its performance in extensive simulation and physical experiments.

This dissertation is composed of three complementary parts. In Part I, we propose novel algorithms and systems for interactive imitation learning, in which autonomous robots can actively query human supervisors for assistance when needed. In Part II, we introduce interactive fleet learning, which generalizes interactive imitation learning to multiple robots and multiple human supervisors. In Part III, we introduce and study systems for remote supervision of robot fleets over the Internet, enabling interactive fleet learning at a distance. Throughout this thesis, we design algorithms and systems with an emphasis on scalability in terms of the number of robots, number of humans, amount of human supervision required, dataset size, and distribution of physical locations. We conclude with a discussion of limitations and opportunities for future work.

To my parents, and to all of my teachers.

Contents

Contents **ii**

1 Introduction **1**
1.1 Scalable Interactive Imitation Learning 2
1.2 Interactive Fleet Learning 3
1.3 Systems for Remote Fleet Supervision 3
1.4 Related Work 4
1.5 Thesis Contributions 6

I Scalable Interactive Imitation Learning **7**

2 LazyDAgger: Reducing Context Switching **8**
2.1 Introduction 8
2.2 Background and Related Work 10
2.3 Problem Statement 11
2.4 Preliminaries: SafeDAgger 12
2.5 LazyDAgger 13
2.6 Experiments 16
2.7 Discussion and Future Work 21

3 ThriftyDAgger: Budget-Aware Novelty and Risk **22**
3.1 Introduction 22
3.2 Related Work 24
3.3 Problem Statement 25
3.4 ThriftyDAgger 26
3.5 Experiments 29
3.6 Discussion and Future Work 33

4 IntervenGen: Interventional Data Generation **35**
4.1 Introduction 35
4.2 Related Work 37

4.3 Preliminaries . . . 39
4.4 IntervenGen . . . 40
4.5 Experiment Setup . . . 42
4.6 Experiments . . . 45
4.7 Conclusion . . . 48

II Interactive Fleet Learning 50

5 Fleet-DAgger: Interactive Robot Fleet Learning 51
5.1 Introduction . . . 51
5.2 Related Work . . . 52
5.3 Interactive Fleet Learning Problem Formulation . . . 54
5.4 Interactive Fleet Learning Algorithms . . . 55
5.5 Interactive Fleet Learning Benchmark . . . 57
5.6 Experiments . . . 59
5.7 Limitations and Future Work . . . 62

6 IIFL: Implicit Interactive Fleet Learning 63
6.1 Introduction . . . 63
6.2 Preliminaries and Related Work . . . 64
6.3 Problem Statement . . . 67
6.4 Approach . . . 68
6.5 Experiments . . . 69
6.6 Limitations and Future Work . . . 73

III Systems for Remote Fleet Supervision 75

7 Real-Time Remote Robot Manipulation 76
7.1 Introduction . . . 76
7.2 Related Work . . . 77
7.3 The Google Reach Testbed . . . 78
7.4 Garment Folding Algorithms . . . 81
7.5 Experiments . . . 85
7.6 Conclusion and Future Work . . . 89

8 FogROS2-SGC: Cloud Robotics with Secure Global Connectivity 91
8.1 Introduction . . . 91
8.2 Related Work . . . 94
8.3 Ten FogROS2-SGC Features . . . 95
8.4 FogROS2-SGC Design . . . 97
8.5 Evaluation . . . 102

8.6 Conclusions and Limitations 107

IV Conclusion **108**

9 Conclusion **109**
9.1 Summary 109
9.2 Limitations and Opportunities for Future Work 110
9.3 Broader Perspective on Robot Learning 112

Bibliography **113**

V Appendices **137**

A Appendix for Chapter 2 **138**
A.1 MuJoCo 138
A.2 LazyDAgger Switching Thresholds 139
A.3 Fabric Smoothing in Simulation 139
A.4 Fabric Manipulation with the ABB YuMi 144

B Appendix for Chapter 3 **147**
B.1 Algorithm Details 147
B.2 Hyperparameter and Implementation Details 150
B.3 Environment Details 152
B.4 User Study Details 152

C Appendix for Chapter 5 **155**
C.1 Mathematical Details of the IFL Problem Formulation 155
C.2 Fleet-DAgger Algorithm Details 156
C.3 Additional Experiment Details 157
C.4 Hyperparameter Sensitivity and Ablation Studies 159

D Appendix for Chapter 6 **165**
D.1 Jeffreys Divergence Identity 165
D.2 Additional Details on Implicit Models 165
D.3 Uncertainty Estimation with Larger Ensembles 166
D.4 Additional Experimental Details 169

E Appendix for Chapter 7 **173**
E.1 Flattening Algorithm Details 173
E.2 Folding Algorithm Details 178
E.3 Graphical User Interface 179

E.4 Action Primitive Details 179
E.5 Ablation Studies 184

Acknowledgments

This year marks my eighth year in a row at the University of California, Berkeley in the Electrical Engineering and Computer Sciences (EECS) department. I began my studies here as a wide-eyed freshman undergraduate in 2016 and entered the Ph.D. program in 2020. My time here has been a transformative one of intense personal growth, learning, and fun, and I owe many individuals a debt of gratitude.

First, to the EECS faculty and staff. I have always had a love of learning throughout my life, and here I've been able to indulge it to the fullest extent. I've taken dozens of courses over the years, each one more fascinating than the last. The EECS curriculum is truly exceptional, with an attention to detail and rigor that I deeply admire as well as dedicated and inspiring instructors. The amount of work that happens behind the scenes to offer these courses so smoothly is something I had taken for granted until I got the chance to serve on the other side of things as a graduate student instructor near the end of my Ph.D. Thanks especially to my committee members Prof. Anca Dragan and Prof. Pieter Abbeel, whose teaching of CS 188 and CS 287 respectively first piqued my interest in robotics and AI. Thanks also to EECS staff (especially Shirley Salanio and Jean Nguyen) and BAIR staff (especially Angie Abbatecola), who have been consistently helpful throughout my Ph.D.

My path into research and a Ph.D. was much more of a serendipitous accident than a premeditated effort. I owe a great deal to three people in particular. First is Prof. Daniel Seita, who was my research mentor when I first joined the AUTOLab as a junior in 2018 and is now a professor at USC. I came in knowing virtually nothing about robotics research, and Daniel patiently mentored me over the course of two years as we worked closely on several research projects. Research is challenging, and after a few months I felt that I was in over my head and wasn't cut out for it. Daniel convinced me to keep going and generously helped me through the research roadblocks I was facing at the time, and since then I have come to love research.

Second is Dr. Ashwin Balakrishna, who was a Ph.D. student in the AUTOLab two years ahead of me. We started working together in 2019, and he then became my closest collaborator during the first two years of my Ph.D. from 2020 to 2022. Much of the research in this thesis grew out of our frequent discussions and would not be possible without him, and he has been a role model for me throughout my Ph.D. I have grown significantly as a researcher under his mentorship and I continue to learn a great deal from our friendship today.

Third is my advisor, Prof. Ken Goldberg, who took a chance on me as an undergraduate eager to prove myself in the lab. It has been an absolute pleasure to be in the AUTOLab for the last six years, and Prof. Goldberg has been one of the biggest mentors in my life. He has given extremely detailed and helpful feedback on multiple iterations of every single manuscript and presentation I've worked on. He has skillfully guided me toward fruitful research directions based on his decades of experience while simultaneously giving me the freedom to deeply pursue my interest in imitation learning. He is also a perpetual champion

of my work and has regularly given me invaluable career opportunities throughout my time here. I look forward to fun research discussions for many years to come.

Countless others deserve acknowledgements, but for the sake of brevity I'll highlight only a small fraction. To my senior collaborators, especially Prof. Daniel Brown, Dr. Ellen Novoseller, and Dr. Brijen Thananjeyan: thank you for all your valuable advice and mentorship, and it has been a joy working together over the years. To the undergraduate and Master's students I've had the opportunity to mentor and work with, especially Satvik Sharma, Gaurav Datta, Kaushik Shivakumar, and Karthik Dharmarajan: I have been consistently impressed by your talent, and it has been very rewarding to watch you become skilled researchers. To my collaborators at NVIDIA during my recent internship there, especially Dr. Ajay Mandlekar: thank you for your consistent generosity with your time and valuable perspective on imitation learning; I feel that as a result I have grown significantly as a researcher in a short time. To my Ph.D. student peers in my lab and others, especially Justin Kerr and Rohan Taori: your research trajectories have been very inspiring to watch and I've learned a lot from our frequent discussions and friendship. Finally, to my family and friends, especially Chester Leung, Rohen Sukkawala, Aditya Ganapathi, Chanan Walia, Amog Kamsetty, my brother, and my parents: thank you for your consistent support and friendship over the last 10+ years of my life. I would not have had the tenacity and optimism required to finish my Ph.D. without all of you, so it is as much your accomplishment as it is mine.

Chapter 1

Introduction

Machines that are operated by humans are capable of human-level dexterity [278]. With *teleoperation*, such operation can be performed at a distance, enabling applications from driving a vehicle remotely over the Internet to exploring the surface of Mars. However, fully autonomous robots still struggle with achieving this level of dexterity in the same range of unstructured environments.

Meanwhile, in the last few years, large high-capacity models trained with supervised machine learning on Internet-scale data have achieved dramatic breakthroughs in natural language processing and computer vision [206, 186, 10]. Can we apply a similar supervised learning recipe to robotics, using human control data collected via teleoperation?

This approach presents three major challenges. First, because robots must execute their control policies in physical environments rather than imitate static datasets, they encounter *distribution shift* between the states they were trained on and the states they visit during policy execution. Second, real-world data distributions consist of "long-tail" phenomena: a large variety of low-probability data that is unlikely to appear at training time. Third, teleoperated robot control data is not freely available on the Internet. Robot control data is extremely scarce compared to text and image data, and the data scaling laws are unknown. We have a catch-22: robot learning requires large datasets from deployment at scale, but robot learning is not yet reliable enough to be deployed at scale.

Despite these challenges, large-scale deployments of robot learning systems have begun to emerge in industrial and commercial settings ranging from autonomous driving to e-commerce order fulfillment. How is this possible? A popular approach is to fall back on human teleoperators when robot autonomy is unreliable [28, 240, 154, 50].

Rather than considering robot autonomy as a binary decision variable, where a system is either fully teleoperated or fully autonomous, we consider the setting of *supervised autonomy*, in which systems are partially autonomous and partially teleoperated. With supervised autonomy, an autonomous robot system that encounters an edge case can cede control to a human supervisor. This setting confers several compelling advantages: (1) the degree of autonomy can improve continuously over time; (2) a single human can supervise multiple robots; and (3) the human-robot team can be deployed immediately without full robot

autonomy. Moreover, when deployed with a large robot fleet, more long-tail situations are encountered and each robot can learn from the experience of the other robots. In this dissertation, we focus on this idea of supervised autonomy and how it may be implemented at scale.

This thesis is structured in three parts. In Part I, we propose novel algorithms and systems for implementing supervised autonomy with *interactive imitation learning (IIL)*, in which human supervisors can provide corrective interventions during robot policy execution. In Part II, we explore how scalable IIL algorithms enable the supervision of large robot fleets with a small set of human supervisors (i.e., where the number of robots exceeds the number of humans by 10× or more). Then, in Part III, we study systems that enable human supervision of robot fleets remotely over the Internet.

Putting these components together, we propose a comprehensive approach to supervised autonomy that both facilitates reliable robot deployment in contemporary society and builds a potential path toward increasingly capable robot autonomy. A central emphasis throughout this work is *scalability*: by minimizing the burden on human supervisors, enabling the simultaneous supervision of multiple robots, and allowing supervision to come from anywhere on Earth, we design our algorithms and systems to facilitate the scaling of robot data collection toward the colossal size of modern language and vision datasets.

1.1 Scalable Interactive Imitation Learning

Imitation learning (IL), in which robots learn from human feedback and examples, has become a leading paradigm for training robot control policies. The simplest approach is behavior cloning: first a set of human demonstrations are collected via teleoperation, then a mapping from observations to actions is learned via supervised learning. However, behavior cloning can lead to a mismatch between the state distribution visited by the human and that visited by the robot [215]. To address this, *interactive imitation learning (IIL)* algorithms such as DAgger [215] and its variants [171, 125, 276] have the robot periodically cede control to a human supervisor during policy execution for corrective interventions.

In Part I, we propose novel algorithms and systems for the IIL setting. IIL confers two benefits over behavioral cloning: (1) the human-robot team can provide higher reliability than the autonomous robot alone, and (2) the human intervention data can refine the robot policy via additional supervised learning. Chapters 2 and 3 present LazyDAgger and ThriftyDAgger, respectively: two novel algorithms for *robot-gated* IIL, in which the system autonomously decides when human interventions should occur and actively queries for human assistance. As opposed to human-gated IIL [125], robot-gated algorithms enable "on-demand" supervision: the human supervisor does not have to continuously monitor the system and is only called when necessary. This significantly facilitates scalability by freeing human attention to attend to other tasks (such as helping other robots). In Chapter 4, we further explore scalability with IntervenGen, a data generation system for IIL. IL is notoriously burdensome for human supervisors in terms of data collection time and effort,

and IIL is arguably even more burdensome. But how much of the data actually consists of unique behaviors? From a small budget of only 10 human interventions, IntervenGen can autonomously synthesize 1000 interventions or more, providing broad coverage of the state space without any additional burden on the human supervisor.

1.2 Interactive Fleet Learning

In Part II, we investigate how the on-demand supervision of Part I may facilitate the management of large fleets of robots that significantly outnumber the pool of available human supervisors. Since robot-gated IIL does not require supervisors to actively monitor the robot systems, a single human can flexibly supervise multiple robots, given an effective human-to-robot allocation strategy. In Chapter 5, we propose *interactive fleet learning (IFL)*: the first formalism for interactive IL with multiple robots and multiple humans. We also introduce novel algorithms and benchmarks for the IFL setting as well as large-scale empirical analysis with fleets of 100+ robots in simulation and 4 physical robots. However, with multiple human supervisors, the human teleoperation policy may be multimodal according to individual human preferences and proficiency. Accordingly, in Chapter 6, we propose Implicit IFL, an algorithm that extends IFL by enabling the robot fleet to learn from heterogeneous human demonstrations and interventions. To do so, we propose a novel technique for estimating epistemic uncertainty in energy-based models [141].

1.3 Systems for Remote Fleet Supervision

Lastly, in Part III, we consider systems for remote supervision of robot fleets. Mature Internet and telecommunication technology enables real-time robot teleoperation at a distance [17]. This further facilitates scalability by allowing human supervision to come from anywhere around the globe, regardless of where the robot fleet is operating. Moreover, with cloud and fog robotics [123], robots are no longer limited to on-board compute and memory as they can tap into vast cloud computing resources. Chapter 7 studies Reach [266], a prototype commercial remote robot workcell, and how it can be used to facilitate standardized benchmarking for robotic manipulation. The system enables remote human teleoperation and closed-loop visuomotor policy execution at 10 Hz over the Internet. Then, in Chapter 8, we propose a cloud robotics platform for securely connecting disjoint Robot Operating System (ROS) networks. We show that such a system can facilitate real-time robot fleet operation with robots, compute nodes, and teleoperators distributed around the globe.

1.4 Related Work

Interactive Imitation Learning

Imitation learning is an increasingly popular paradigm for robot learning [15, 253, 54, 203, 95, 12]. However, learning from purely offline data often suffers from distribution shift [215, 138], as compounding approximation error leads to states that were not visited by the human. This can be mitigated with online data collection with interactive imitation learning (IIL) algorithms such as Dataset Aggregration (DAgger) [215] and its variants [42, 110, 211]. Human-gated IIL algorithms [125, 241, 160] require the human to monitor the robot learning process and decide when to take and cede control of the system. While intuitive, these approaches are not scalable to large fleets of robots or the long periods of time involved in continual learning, as humans cannot effectively focus on many robots simultaneously [45, 200, 46] and are prone to fatigue [177].

In contrast, robot-gated IIL algorithms such as EnsembleDAgger [171] and SafeDAgger [276] allow the robot to solicit interventions from a human when the system deems necessary. In practice, these algorithms estimate various quantities correlated with task performance [276, 214] and uncertainty [171, 139] and use them to determine when to solicit interventions. Prior work has proposed intervention criteria which use the novelty of states visited by the robot [171] or the predicted discrepancy between the actions proposed by the robot policy and by the supervisor [276]. However, finding intervention criteria that effectively balance task performance with supervisor burden is very challenging; we propose new methods for improving this balance.

Interactive reinforcement learning (RL) [271, 135, 261, 116, 254, 150] is another active area of research in which robots learn from both online human feedback and their own experience. There has also been recent work that applies conformal prediction [9] to uncertainty-based intervention criteria for robot planning with large language models [210].

Fleet Learning and Management

For human-robot teams, deciding when to transfer control between robots and humans during execution is a widely studied topic in the literature of both sliding autonomy [229, 228, 67] and Human-Robot Interaction (HRI). In sliding autonomy, also known as adjustable autonomy [224, 132] or adaptive automation [234], humans and robots dynamically adjust their level of autonomy and transfer control to each other during execution [67, 234]. Since identifying which robot to assist in a large robot fleet can be overwhelming for a human operator [37, 142, 45, 200, 46], several strategies such as using a cost-benefit analysis to decide whether to request operator assistance [229] and using an advising agent to filter robot requests [213] have been proposed to improve the performance of human-robot teams [200, 213, 57] and increase the number of robots that can be controlled [274], a quantity known as "fan-out" [184]. Other examples include user modeling [229, 228, 200, 57] and studying interaction modes [32] for better system and interface design [6, 36]. Zheng et al. [279]

propose computing the estimated time until stopping for mobile robots and prioritizing robots accordingly. Ji et al. [112] consider the setting where physical assistance is required to resume tasks for navigation robots and formalize single-human, multi-robot allocation as graph traversal. Dahiya et al. [62] formulate the problem of multi-human, multi-robot allocation during execution as a Restless Multi-Armed Bandit problem. Allocation of humans to robots has also been studied from the perspectives of queueing theory and scheduling theory [46, 56, 230, 84, 216, 65]. The vast majority of the human-robot teaming and queueing theory work, however, does not involve learning; the robot control policies are assumed to be fixed. In contrast, we study supervisor allocation during robot learning, where allocation affects not only human burden and task performance but also the efficiency of policy learning.

There have also been several works that involve learning across fleets of multiple robots [109, 27, 94, 117, 3]. These primarily involve scaling data collection across multiple robots for imitation learning [109, 27, 3] or reinforcement learning [117, 118, 94]. However, these works do not study the supervisor allocation problem. Swamy et al. [245] consider multi-robot interactive imitation learning but with a single human operator. In work subsequent to ours, Ahn et al. [3] consider multi-robot, multi-human allocation, but humans are allocated uniformly according to availability rather than to the robots that require the most assistance.

Remote Robot Systems

Goldberg et al. [83] pioneered remote robot control over the Internet in 1995 with the Telegarden, a garden maintained remotely by 10,000 people over 9 years. James Kaufner introduced the term "Cloud Robotics" in 2010 to refer to robot systems that are not limited to on-board compute and data [123]. Cloud and fog computing have been applied to robotic tasks such as grasp planning [248, 122, 143], parallelized Monte-Carlo grasp perturbation sampling [120, 121, 124], and motion planning [136, 41, 104]. Ichnowski et al. [104] propose frameworks for offloading computation to resources on the edge or cloud, while Anand et al. [8] present systems that leverage serverless computing [170].

Modern computing paradigms have enabled new applications such as multi-robot fleet learning [94] and remote sharing of robot systems [246, 17]. Some remote robotics testbeds include Robotarium [202] for swarm robotics and Duckietown [195] for autonomous driving. Mandlekar et al. [163] developed RoboTurk, a system for crowdsourcing of robot data collection via low-latency teleoperation over the Internet. More recently, Bauer et al. [17] hosted the online "Real Robot Challenge" for manipulation in 2020 and 2021 at Neural Information Processing Systems (NeurIPS). Six robotics groups from around the world were able to access their tri-finger robot [269] remotely via the Internet and evaluate their algorithms on the shared infrastructure. In this work, we use a new remote robot testbed for real-time deformable object manipulation with closed-loop visuomotor control, and we introduce a novel system for connecting disjoint robot networks around the globe.

1.5 Thesis Contributions

The primary contributions of this thesis are the following:

- Novel intervention criteria that enable robot systems to actively query for human assistance with minimal burden on human supervisors in Chapters 2, 3, 5, and 6.
- New metrics for evaluating interactive IL including "context switching" in Chapter 2 and "return on human effort" in Chapter 5, as well as the first theoretical formalism for multi-robot, multi-human interactive IL in Chapter 5.
- A system for autonomously generating synthetic interventional data in Chapter 4.
- A new computationally efficient technique for quantifying epistemic uncertainty in energy-based models in Chapter 6.
- Open-source software infrastructure and thorough empirical analysis of human-to-robot allocation with large-scale simulated fleets of 100+ robots in Chapters 5 and 6.
- Systems for real-time remote supervision of physical robot fleets over the Internet in Chapters 5, 6, 7, and 8.

Part I

Scalable Interactive Imitation Learning

Chapter 2

LazyDAgger: Reducing Context Switching

In this chapter, we present LazyDAgger, a novel algorithm for robot-gated interactive imitation learning. Compared to existing baselines, LazyDAgger significantly reduces context switching between robot and human control.

2.1 Introduction

Imitation learning allows a robot to learn from human feedback and examples [12, 13, 187]. In particular, *interactive imitation learning* (IL) [241, 125, 276], in which a human supervisor periodically takes control of the robotic system during policy learning, has emerged as a popular imitation learning method, as interventions are a particularly intuitive form of human feedback [241]. However, a key challenge in interactive imitation learning is to reduce the burden that interventions place on the human supervisor [276, 125].

One source of this burden is the cost of *context switches* between human and robot control. Context switches incur significant time cost, as a human must interrupt the task they are currently performing, acquire control of the robot, and gain sufficient situational awareness before beginning the intervention. As an illustrative example, consider a robot performing a task for which an action takes 1 time unit and an intervention requires two context switches (one at the start and one at the end). We define *latency* L as the number of time units associated with a single context switch. For instance, $L \gg 1$ for a human supervisor who will need to pause an ongoing task and walk over to a robot that requires assistance. If the supervisor takes control 10 times for 2 actions each, she spends $20L + 20$ time units helping the robot. In contrast, if the human takes control 2 times for 10 actions each, she spends only $4L + 20$ time units. The latter significantly reduces the burden on the supervisor. Furthermore, prior work suggests that frequent context switches can make it difficult for the supervisor to perform other tasks in parallel [245] or gain enough situational awareness to provide useful interventions [209].

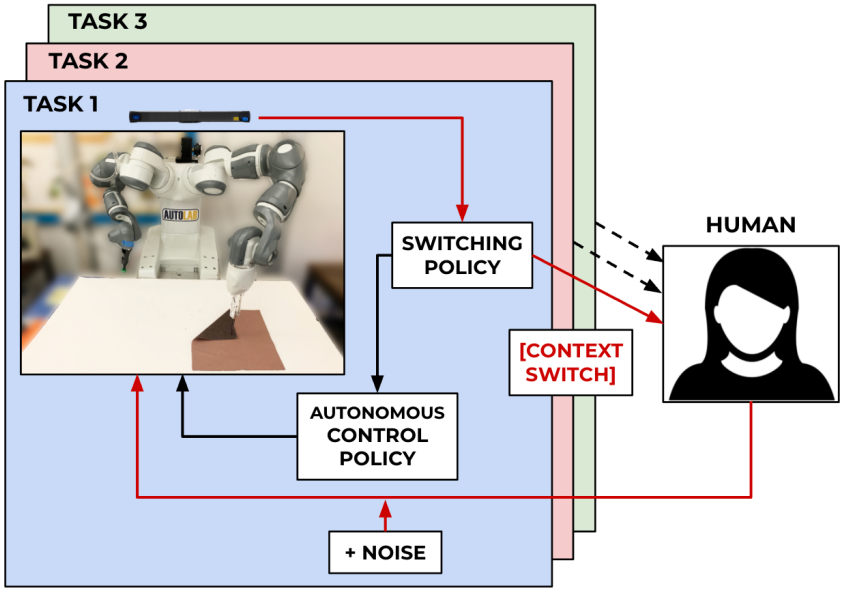


Figure 2.1: LazyDAgger learns to cede control to a supervisor in states in which it estimates that its actions will significantly deviate from those of the supervisor. LazyDAgger reduces context switches between supervisor and autonomous control to reduce burden on a human supervisor working on multiple tasks.

We present LazyDAgger (Figure 2.1), an algorithm which initiates useful interventions while limiting context switches. The name LazyDAgger is inspired by the concept of lazy evaluation in programming language theory [140], where expressions are evaluated only when required to reduce computational burden. As in SafeDAgger [276], LazyDAgger learns a meta-controller which determines when to context switch based on the estimated discrepancy between the learner and supervisor. However, unlike SafeDAgger, LazyDAgger reduces context switching by (1) introducing asymmetric switching criteria and (2) injecting noise into the supervisor control actions to widen the distribution of visited states. One appealing property of this improved meta-controller is that even after training, LazyDAgger can be applied at execution time to improve the safety and reliability of autonomous policies with minimal context switching. We find that across 3 continuous control tasks in simulation, LazyDAgger achieves task performance on par with DAgger [215] with 88% fewer supervisor actions than DAgger and 60% fewer context switches than SafeDAgger. In physical fabric manipulation experiments, we observe similar results, and find that at execution time, LazyDAgger achieves 60% better task performance than SafeDAgger with 60% fewer context switches.

2.2 Background and Related Work

Challenges in learning efficiency and reward function specification have inspired significant interest in algorithms that can leverage supervisor demonstrations and feedback for policy learning.

Learning from Offline Demonstrations: Learning from demonstrations [12, 187, 13] is a popular imitation learning approach, as it requires minimal supervisor burden: the supervisor provides a batch of offline demonstrations and gives no further input during policy learning. Many methods use demonstrations directly for policy learning [203, 106, 194, 253], while others use reinforcement learning to train a policy using a reward function inferred from demonstrations [1, 281, 95, 29, 77]. Recent work has augmented demonstrations with additional offline information such as pairwise preferences [31, 30], human gaze [221], and natural language descriptions [255]. While offline demonstrations are often simple to provide, the lack of online feedback makes it difficult to address specific bottlenecks in the learning process or errors in the resulting policy due to covariate shift [215].

Learning from Online Feedback: Many policy learning algorithms' poor performance stems from a lack of online supervisor guidance, motivating active learning methods such as DAgger, which queries the supervisor for an action in every state that the learner visits [215]. While DAgger has a number of desirable theoretical properties, labeling every state is costly in human time and can be a non-intuitive form of human feedback [138]. More generally, the idea of learning from action advice has been widely explored in imitation learning algorithms [16, 139, 115, 110]. There has also been significant recent interest in active preference queries for learning reward functions from pairwise preferences over demonstrations [220, 49, 102, 190, 21, 30]. However, many forms of human advice can be unintuitive, since the learner may visit states that are significantly far from those the human supervisor would visit, making it difficult for humans to judge what correct behavior looks like without interacting with the environment themselves [241, 208].

Learning from Supervisor Interventions: There has been significant prior work on algorithms for learning policies from interventions. Xie et al. [271] and Kurenkov et al. [135] leverage interventions from suboptimal supervisors to accelerate policy learning, but assume that the supervisors are algorithmic and thus can be queried cheaply. Thananjeyan et al. [247], Wagener et al. [260], and Saunders et al. [222] also leverage interventions from algorithmic policies, but for constraint satisfaction during learning. Kelly et al. [125], Spencer et al. [241], Wang et al. [261], Kahn et al. [116], Mandlekar et al. [160], and Amir et al. [7] instead consider learning from human supervisors and present learning algorithms which utilize the timing and nature of human interventions to update the learned policy. By giving the human control for multiple timesteps in a row, these algorithms show improvements over methods that only hand over control on a state-by-state basis [18]. However, the above algorithms assume that the human is continuously monitoring the system to determine when to intervene, which may not be practical in large-scale systems or continuous learning settings [58, 36, 245, 126]. Such algorithms also assume that the human knows when to cede control to the robot, which requires guessing how the robot will behave in the future. Zhang et al.

[276] and Menda et al. [171] present imitation learning algorithms SafeDAgger and EnsembleDAgger, respectively, to address these issues by learning to request interventions from a supervisor based on measures such as state novelty or estimated discrepancy between the learner and supervisor actions. These methods can still be sample inefficient, and, as we discuss later, often result in significant context switching. By contrast, LazyDAgger encourages interventions that are both easier to provide and more informative. To do this, LazyDAgger prioritizes (1) sustained interventions, which allow the supervisor to act over a small number of contiguous sequences of states rather than a large number of disconnected intervals, and (2) interventions which demonstrate supervisor actions in novel states to increase robustness to covariate shift in the learned policy.

2.3 Problem Statement

We consider a setting in which a human supervisor is training a robot to reliably perform a task. The robot may query the human for assistance, upon which the supervisor takes control and teleoperates the robot until the system determines that it no longer needs assistance. We assume that the robot and human policy have the same action space, and that it is possible to pause task execution while waiting to transfer control. We formalize these ideas in the context of prior imitation learning literature.

We model the environment as a discrete-time Markov decision process (MDP) $\mathcal{M}$ with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, and time horizon T [205]. The robot does not have access to the reward function or transition dynamics of $\mathcal{M}$ but can cede control to a human supervisor, who executes some deterministic policy $\pi_H : \mathcal{S} \to \mathcal{A}$. We refer to times when the robot is in control as *autonomous mode* and those in which the supervisor is in control as *supervisor mode*. We minimize a surrogate loss function $J(\pi_R)$ to encourage the robot policy $\pi_R : \mathcal{S} \to \mathcal{A}$ to match that of the supervisor (π_H):

$$J(\pi_R) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi_R}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right], \tag{2.1}$$

where $\mathcal{L}(\pi_R(s), \pi_H(s))$ is an action discrepancy measure between $\pi_R(s)$ and $\pi_H(s)$ (e.g., the squared loss or 0-1 loss), and $d_t^{\pi_R}$ is the marginal state distribution at timestep t induced by executing π_R in MDP $\mathcal{M}$.

In interactive IL we require a meta-controller π that determines whether to query the robot policy π_R or to query for an intervention from the human supervisor policy π_H; importantly, π consists of both (1) the high-level controller which decides whether to switch between π_R and π_H and (2) the low-level robot policy π_R. A key objective in interactive IL is to minimize some notion of supervisor burden. To this end, let $m_I(s_t; \pi)$ be an indicator which records whether a context switch between autonomous (π_R) and supervisor (π_H) modes occurs at state s_t (either direction). Then, we define $C(\pi)$, the expected number of context switches in an episode under policy π, as follows: $C(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi}} \left[m_I(s_t; \pi) \right]$, where d_t^{π}

is the marginal state distribution at timestep t induced by executing the meta-controller π in MDP $\mathcal{M}$. Similarly, let $m_H(s_t;\pi)$ indicate whether the system is in supervisor mode at state s_t. We then define $D(\pi)$, the expected number of supervisor actions in an episode for the policy π, as follows: $D(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi}} [m_H(s_t;\pi)]$.

We define *supervisor burden* $B(\pi)$ as the expected time cost imposed on the human supervisor. This can be expressed as the sum of the expected total number of time units spent in context switching and the expected total number of time units in which the supervisor is actually engaged in performing interventions:

$$B(\pi) = L \cdot C(\pi) + D(\pi), \tag{2.2}$$

where L is context switch latency (Section 7.1) in time units, and each time unit is the time it takes for the supervisor to execute a single action. The learning objective is to find a policy π that matches supervisor performance, π_H, while limiting supervisor burden to lie within a threshold Γ_b, set by the supervisor to an acceptable tolerance for a given task. To formalize this problem, we propose the following objective:

$$\pi = \arg\min_{\pi' \in \Pi} \{ J(\pi'_R) \mid B(\pi') \leq \Gamma_b \}, \tag{2.3}$$

where Π is the space of all meta-controllers, and π'_R is the low-level robot policy associated with meta-controller π'.

2.4 Preliminaries: SafeDAgger

We consider interactive IL in the context of the objective introduced in Equation (2.3): to maximize task reward while limiting supervisor burden. To do this, LazyDAgger builds on SafeDAgger [276], a state-of-the-art algorithm for interactive IL. SafeDAgger selects between autonomous mode and supervisor mode by training a binary action discrepancy classifier f to discriminate between "safe" states which have an action discrepancy below a threshold β_H (i.e., states with $\mathcal{L}(\pi_R(s), \pi_H(s)) < \beta_H$) and "unsafe" states (i.e. states with $\mathcal{L}(\pi_R(s), \pi_H(s)) \geq \beta_H$). The classifier f is a neural network with a sigmoid output layer (i.e., $f(s) \in [0, 1]$) that is trained to minimize binary cross-entropy (BCE) loss on the datapoints $(s_t, \pi_H(s_t))$ sampled from a dataset $\mathcal{D}$ of trajectories collected from π_H. This is written as follows:

$$\begin{aligned} \mathcal{L}_{\mathrm{S}}(\pi_R(s_t), \pi_H(s_t), f) = &-f^*(\pi_R(s_t), \pi_H(s_t)) \log f(s_t) \\ &-(1 - f^*(\pi_R(s_t), \pi_H(s_t))) \log(1 - f(s_t)), \end{aligned} \tag{2.4}$$

where the training labels are given by $f^*(\pi_R(s_t), \pi_H(s_t)) = \mathbb{1}\{\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \geq \beta_H\}$, and $\mathbb{1}$ denotes the indicator function. Thus, $\mathcal{L}_{\mathrm{S}}(\pi_R(s_t), \pi_H(s_t), f)$ penalizes incorrectly classifying a "safe" state as "unsafe" and vice versa.

SafeDAgger executes the meta-policy π which selects between π_R and π_H as follows:

$$\pi(s_t) = \begin{cases} \pi_R(s_t) \text{ if } f(s_t) < 0.5 \\ \pi_H(s_t) \text{ otherwise,} \end{cases} \tag{2.5}$$

where $f(s_t) < 0.5$ corresponds to a prediction that $\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) < \beta_H$, i.e., that s_t is "safe." Intuitively, SafeDAgger only solicits supervisor actions when f predicts that the action discrepancy between π_R and π_H exceeds the safety threshold β_H. Thus, SafeDAgger provides a mechanism for querying the supervisor for interventions only when necessary. In LazyDAgger, we utilize this same mechanism to query for interventions but enforce new properties once we enter these interventions to lengthen them and increase the diversity of states observed during the interventions.

2.5 LazyDAgger

We summarize LazyDAgger in Algorithm 1. In the initial phase (Lines 1-3), we train π_R and safety classifier f on offline datasets collected from the supervisor policy π_H. In the interactive learning phase (Lines 4-19), we evaluate and update the robot policy for N epochs, ceding control to the supervisor when the robot predicts a high action discrepancy.

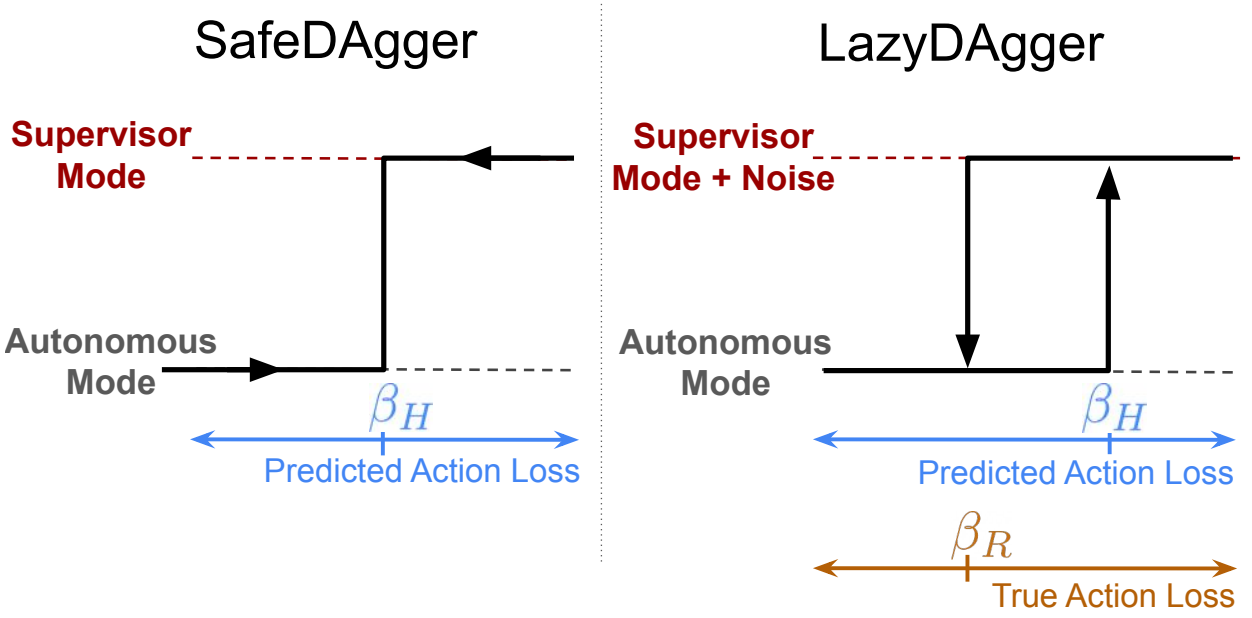


Figure 2.2: **LazyDAgger Switching Strategy:** SafeDAgger switches between supervisor and autonomous mode if the predicted action discrepancy is above threshold β_H. In contrast, LazyDAgger uses asymmetric switching criteria and switches to autonomous mode based on ground truth action discrepancy. The gap between β_R and β_H defines a hysteresis band [23].

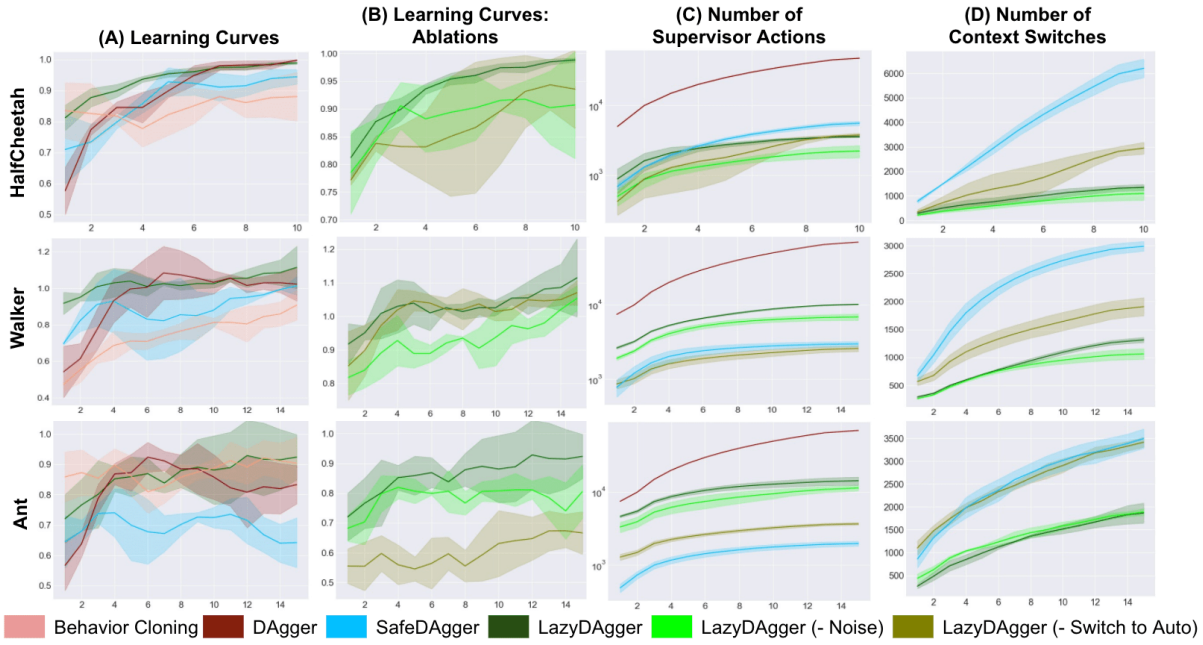


Figure 2.3: **MuJoCo Simulation Results:** We study task performance (A), ablations (B), online supervisor burden (C), and total bidirectional context switches (D) for LazyDAgger and baselines over 3 random seeds. For Columns (A)-(D), the x-axis for all plots shows the number of epochs over the training dataset, while the y-axes indicate normalized reward (A, B), counts of supervisor actions (C, log scale), and context switches (D) with shading for 1 standard deviation. We find that LazyDAgger outperforms all baselines and ablations, indicating that encouraging lengthy, noisy interventions improves performance. Additionally, LazyDAgger uses far fewer context switches than other baselines while requesting far fewer supervisor actions than DAgger.

Action Discrepancy Prediction

SafeDAgger uses the classifier f to select between π_R and π_H (Equation (2.5)). However, in practice, this often leads to frequent context switching (Figure 2.3). To mitigate this, we make two observations. First, we can leverage that in supervisor mode, we directly observe the supervisor's actions. Thus, there is no need to use f, which may have approximation errors, to determine whether to remain in supervisor mode; instead, we can compute the ground-truth action discrepancy $\mathcal{L}(\pi_R(s_t), \pi_H(s_t))$ exactly for any state s_t visited in supervisor mode by comparing the supplied supervisor action $\pi_H(s_t)$ with the action proposed by the robot policy $\pi_R(s_t)$. In contrast, SafeDAgger uses f to determine when to switch modes both in autonomous and supervisor mode, which can lead to very short interventions when f prematurely predicts that the agent can match the supervisor's actions. Second, to ensure the robot has returned to the supervisor's distribution, the robot should only switch

Algorithm 1 LazyDAgger

Require: Number of epochs N, time steps per epoch T, intervention thresholds β_H, β_R, supervisor policy π_H, noise σ^2

1: Collect $\mathcal{D}, \mathcal{D}_{\mathrm{S}}$ offline with supervisor policy π_H
2: $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t, \pi_H(s_t)) \sim \mathcal{D}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right]$ ▷ Eq. (2.1)
3: $f \leftarrow \arg\min_f \mathbb{E}_{(s_t, \pi_H(s_t)) \sim \mathcal{D} \cup \mathcal{D}_{\mathrm{S}}} \left[\mathcal{L}_{\mathrm{S}}(\pi_R(s_t), \pi_H(s_t), f) \right]$ ▷ Eq. (2.4)
4: **for** $i \in \{1, \dots N\}$ **do**
5: Initialize s_0, Mode $\leftarrow$ Autonomous
6: **for** $t \in \{1, \dots T\}$ **do**
7: $a_t \sim \pi_R(s_t)$
8: **if** Mode = Supervisor or $f(s_t) \geq 0.5$ **then**
9: $a_t^H = \pi_H(s_t)$
10: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t^H)\}$
11: Execute $\tilde{a}_t^H \sim \mathcal{N}(a_t^H, \sigma^2 I)$
12: **if** $\mathcal{L}(a_t, a_t^H) < \beta_R$ **then**
13: Mode $\leftarrow$ Autonomous
14: **else**
15: Mode $\leftarrow$ Supervisor
16: **else**
17: Execute a_t
18: $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t, \pi_H(s_t)) \sim \mathcal{D}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right]$
19: $f \leftarrow \arg\min_f \mathbb{E}_{(s_t, \pi_H(s_t)) \sim \mathcal{D} \cup \mathcal{D}_{\mathrm{S}}} \left[\mathcal{L}_{\mathrm{S}}(\pi_R(s_t), \pi_H(s_t), f) \right]$

back to autonomous mode when the action discrepancy falls below a threshold β_R, where $\beta_R < \beta_H$. As illustrated in Figure 2.2, LazyDAgger's asymmetric switching criteria create a hysteresis band, as is often utilized in control theory [23]. Motivated by Eq. (2.3), we adjust β_H to reduce context switches $C(\pi)$ and adjust β_R as a function of β_H to increase intervention length. We hypothesize that redistributing the supervisor actions into fewer but longer sequences in this fashion both reduces burden on the supervisor and improves the quality of the online feedback for the robot. Details on setting these hyperparameter values in practice, the settings used in our experiments, and a hyperparameter sensitivity analysis are provided in the Appendix.

Noise Injection

If the safety classifier is querying for interventions at state s_t, then the robot either does not have much experience in the neighborhood of s_t or has trouble matching the demonstrations at s_t. This motivates exploring novel states near s_t so that the robot can receive maximal feedback on the correct behavior in areas of the state space where it predicts a large action discrepancy from the supervisor. Inspired by prior work that has identified noise injection as a useful tool for improving the performance of imitation learning algorithms (e.g. Laskey et al. [138] and Brown et al. [29]), we diversify the set of states visited in supervisor mode by injecting isotropic Gaussian noise into the supervisor's actions, where the variance σ^2 is a scalar hyperparameter (Line 11 in Algorithm 1).

2.6 Experiments

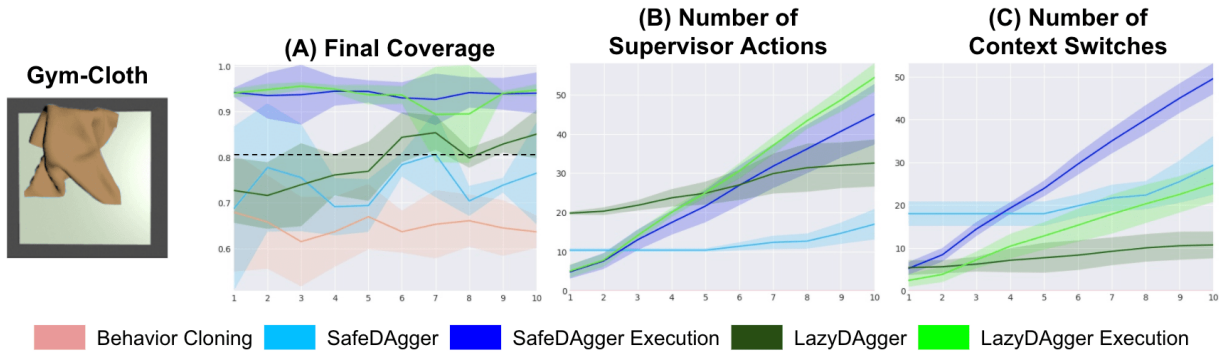


Figure 2.4: **Fabric Smoothing Simulation Results:** We study task performance measured by final fabric coverage (A), total supervisor actions (B), and total context switches (C) for LazyDAgger and baselines in the Gym-Cloth environment from [227]. The horizontal dotted line shows the success threshold for fabric smoothing. LazyDAgger achieves higher final coverage than Behavior Cloning and SafeDAgger with fewer context switches than SafeDAgger but more supervisor actions. At execution time, we again observe that LazyDAgger achieves similar coverage as SafeDAgger but with fewer context switches.

We study whether LazyDAgger can (1) reduce supervisor burden while (2) achieving similar or superior task performance compared to prior algorithms. Implementation details are provided in the supplementary material. In all experiments, $\mathcal{L}$ measures Euclidean distance.

Simulation Experiments: MuJoCo Benchmarks

Environments: We evaluate LazyDAgger and baselines on 3 continuous control environments from MuJoCo [251], a standard simulator for evaluating imitation and reinforcement learning algorithms. In particular, we evaluate on HalfCheetah-v2, Walker2D-v2 and Ant-v2.

Metrics: For LazyDAgger and all baselines, we report learning curves which indicate how quickly they can make task progress in addition to metrics regarding the burden imposed on the supervisor. To study supervisor burden, we report the number of supervisor actions, the number of context switches, and the total supervisor burden (as defined in Eq. (2.2)). Additionally, we define $L^* \geq 0$ to be the latency value such that for all $L > L^*$, LazyDAgger has a lower supervisor burden than SafeDAgger. We report this L^* value, which we refer to as the *cutoff latency*, for all experiments to precisely study the types of domains in which LazyDAgger is most applicable.

Baselines: We compare LazyDAgger to Behavior Cloning [253], DAgger [215], and SafeDAgger [276] in terms of the total supervisor burden and task performance. The Be-

havior Cloning and DAgger comparisons evaluate the utility of human interventions, while the comparison to SafeDAgger, another interactive IL algorithm, evaluates the impact of soliciting fewer but longer interventions.

Experimental Setup: For all MuJoCo environments, we use a reinforcement learning agent trained with TD3 [78] as an algorithmic supervisor. We begin all LazyDAgger, SafeDAgger, and DAgger experiments by pre-training the robot policy with Behavior Cloning on 4,000 state-action pairs for 5 epochs, and similarly report results for Behavior Cloning after the 5th epoch. To ensure a fair comparison, Behavior Cloning uses additional offline data equal to the average amount of online data seen by LazyDAgger during training. All results are averaged over 3 random seeds.

Results: In Figure 2.3, we study the performance of LazyDAgger and baselines. After every epoch of training, we run the policy for 10 test rollouts *where interventions are not allowed* and report the task reward on these rollouts in Figure 2.3. Results suggest that LazyDAgger is able to match or outperform all baselines in terms of task performance across all simulation environments (Figure 2.3A). Additionally, LazyDAgger requires far fewer context switches compared to SafeDAgger (Figure 2.3D), while requesting a similar number of supervisor actions across domains (Figure 2.3C): we observe a 79%, 56%, and 46% reduction in context switches on the HalfCheetah, Walker2D, and Ant environments respectively. LazyDAgger and SafeDAgger both use an order of magnitude fewer supervisor actions than DAgger. While SafeDAgger requests much fewer supervisor actions than LazyDAgger in the Ant environment, this limited amount of supervision is insufficient to match the task performance of LazyDAgger or any of the baselines, suggesting that SafeDAgger may be terminating interventions prematurely. We study the total supervisor burden of SafeDAgger and LazyDAgger as defined in Equation (2.2) and find that in HalfCheetah, Walker2D, and Ant, the cutoff latencies L^* are 0.0, 4.3, and 7.6 respectively, i.e. LazyDAgger achieves lower supervisor burden in the HalfCheetah domain for any L as well as lower burden in Walker2D and Ant for $L > 4.3$ and $L > 7.6$ respectively. The results suggest that LazyDAgger can reduce total supervisor burden compared to SafeDAgger even for modest latency values, but that SafeDAgger may be a better option for settings with extremely low latency.

Ablations: We study 2 key ablations for LazyDAgger in simulation: (1) returning to autonomous mode with $f(\cdot)$ rather than using the ground truth discrepancy (LazyDAgger (-Switch to Auto) in Figure 2.3), and (2) removal of noise injection (LazyDAgger (-Noise)). LazyDAgger outperforms both ablations on all tasks, with the exception of ablation 1 on Walker2D, which performed similarly well. We also observe that LazyDAgger consistently requests more supervisor actions than either ablation. This aligns with the intuition that both using the ground truth action discrepancy to switch back to autonomous mode and injecting noise result in longer but more useful interventions that improve performance.

Fabric Smoothing in Simulation

Environment: We evaluate LazyDAgger on the fabric smoothing task from [227] (shown in Figure 2.4) using the simulation environment from [227]. The task requires smoothing an

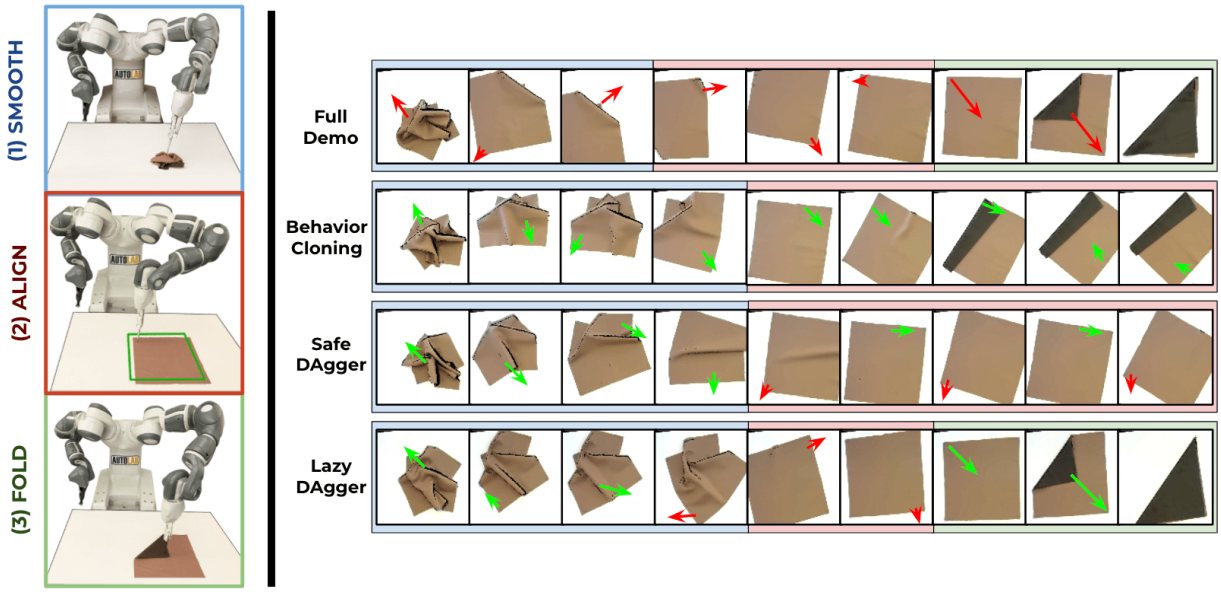


Figure 2.5: **Physical Fabric Manipulation Task:** *Left:* We evaluate on a 3-stage fabric manipulation task consisting of smoothing a crumpled fabric, aligning the fabric so all corners are visible in the observations, and performing a triangular fold. *Right:* Rollouts of the fabric manipulation task, where each frame is a $100 \times 100 \times 3$ overhead image. Human supervisor actions are denoted in red while autonomous robot actions are in green. Rollouts are shaded to indicate task progress: blue for smoothing, red for alignment, and green for folding. SafeDAgger ends human intervention prematurely, resulting in poor task performance and more context switches, while LazyDAgger switches back to robot control only when confident in task completion.

initially crumpled fabric and is challenging due to the infinite-dimensional state space and complex dynamics, motivating learning from human feedback. As in prior work [227], we utilize top-down $100 \times 100 \times 3$ RGB image observations of the workspace and use actions which consist of a 2D pick point and a 2D pull vector. See [227] for further details on the fabric simulator.

Experimental Setup: We train a fabric smoothing policy in simulation using DAgger under supervision from an analytic corner-pulling policy that leverages the simulator's state to identify fabric corners, iterate through them, and pull them towards the corners of the workspace [227]. We transfer the resulting policy for a 16×16 grid of fabric into a new simulation environment with altered fabric dynamics (i.e. lower spring constant, altered fabric colors, and a higher-fidelity 25×25 discretization) and evaluate LazyDAgger and baselines on how rapidly they can adapt the initial policy to the new domain. As in [227], we terminate rollouts when we exceed 10 time steps, 92% coverage, or have moved the fabric more than 20% out of bounds. We evaluate performance based on a coverage metric, which measures

Algorithm	Task Successes	Task Progress			Context Switches	Supervisor Actions	Robot Actions	Failure Modes			
		(1)	(2)	(3)				A	B	C	D
Behavior Cloning	0/10	6/10	0/10	0/10	N/A	N/A	119	2	1	7	0
SD-Execution	2/10	6/10	4/10	2/10	53	**34**	108	5	0	0	3
LD-Execution	**8/10**	10/10	10/10	8/10	**21**	43	47	0	0	0	2

Table 2.1: **Physical Fabric Manipulation Experiments:** We evaluate LazyDAgger-Execution and baselines on a physical 3-stage fabric manipulation task and report the success rate and supervisor burden in terms of total supervisor actions and bidirectional context switches (summed across all 10 trials). Task Progress indicates how many trials completed each of the 3 stages: Smoothing, Aligning, and Folding. LazyDAgger-Execution achieves more successes with fewer context switches ($L^* = 0.28$). We observe the following failure modes (Table 2.1): (A) action limit hit (> 15 total actions), (B) fabric is more than 50% out of bounds, (C) incorrect predicted pick point, and (D) the policy failed to request an intervention despite high ground truth action discrepancy.

the percentage of the background plane that the fabric covers (fully smooth corresponds to a coverage of 100).

Results: We report results for the fabric smoothing simulation experiments in Figure 2.4. Figure 2.4 (A) shows the performance of the SafeDAgger and LazyDAgger policies during learning. To generate this plot we periodically evaluated each policy on *test* rollouts without interventions. Figure 2.4 (B) and (C) show the number of supervisor actions and context switches required during learning; LazyDAgger performs fewer context switches than SafeDAgger but requires more supervisor actions as the interventions are longer. Results suggest that the cutoff latency (as defined in Section 2.6) is $L^* = 1.5$ for fabric smoothing. Despite fewer context switches, LazyDAgger achieves comparable performance to SafeDAgger, suggesting that LazyDAgger can learn complex, high-dimensional robotic control policies while reducing the number of hand-offs to a supervisor. We also evaluate LazyDAgger-Execution and SafeDAgger-Execution, in which interventions are allowed but the policy is no longer updated (see Section 2.6). We see that in this case, LazyDAgger achieves similar final coverage as SafeDAgger with significantly fewer context switches.

Physical Fabric Manipulation Experiments

Environment: In physical experiments, we evaluate on a multi-stage fabric manipulation task with an ABB YuMi robot and a human supervisor (Figure 2.5). Starting from a crumpled initial fabric state, the task consists of 3 stages: (1) fully smooth the fabric, (2) align the fabric corners with a tight crop of the workspace, and (3) fold the fabric into a triangular fold. Stage (2) in particular requires high precision, motivating human interventions. As in the fabric simulation experiments, we use top-down $100 \times 100 \times 3$ RGB image observations of the workspace and have 4D actions consisting of a pick point and pull vector. The actions are converted to workspace coordinates with a standard calibration

procedure and analytically mapped to the nearest point on the fabric. Human supervisor actions are provided through a point-and-click interface for specifying pick-and-place actions. See the supplement for further details.

Experimental Setup: Here we study how interventions can be leveraged to improve the final task performance even at *execution time*, in which policies are no longer being updated. We collect 20 offline task demonstrations and train an initial policy with behavior cloning. To prevent overfitting to a small amount of real data, we use standard data augmentation techniques such as rotating, scaling, changing brightness, and adding noise to create 10 times as many training examples. We then evaluate the behavior cloning agent (Behavior Cloning) and agents which use the SafeDAgger and LazyDAgger intervention criteria but do not update the policy with new experience or inject noise (SafeDAgger-Execution and LazyDAgger-Execution respectively). We terminate rollouts if the fabric has successfully reached the goal state of the final stage (i.e. forms a perfect or near-perfect dark brown right triangle as in Hoque et al. [100]; see Figure 2.5), more than 50% of the fabric mask is out of view in the current observation, the predicted pick point misses the fabric mask by approximately 50% of the plane or more, or 15 total actions have been executed (either autonomous or supervisor).

Results: We perform 10 physical trials of each technique. In Table 2.1, we report both the overall task success rate and success rates for each of the three stages of the task: (1) Smoothing, (2) Alignment, and (3) Folding. We also report the total number of context switches, supervisor actions, and autonomous robot actions summed across all 10 trials for each algorithm (Behavior Cloning, SafeDAgger-Execution, LazyDAgger-Execution). In Figure 2.5 we provide representative rollouts for each algorithm. Results suggest that Behavior Cloning is insufficient for successfully completing the alignment stage with the required level of precision. SafeDAgger-Execution does not improve the task success rate significantly due to its inability to collect interventions long enough to navigate bottleneck regions in the task (Figure 2.5). LazyDAgger-Execution, however, achieves a much higher success rate than SafeDAgger-Execution and Behavior Cloning with far fewer context switches than SafeDAgger-Execution: LazyDAgger-Execution requests 2.1 context switches on average per trial (i.e. 1.05 interventions) as opposed to 5.3 switches (i.e. 2.65 interventions). LazyDAgger-Execution trials also make far more task progress than the baselines, as all 10 trials reach the folding stage. LazyDAgger-Execution does request more supervisor actions than SafeDAgger-Execution, as in the simulation environments. LazyDAgger-Execution also requests more supervisor actions relative to the total amount of actions due to the more conservative switching criteria and the fact that successful episodes are shorter than unsuccessful episodes on average. Nevertheless, results suggest that for this task, LazyDAgger-Execution reduces supervisor burden for any $L > L^* = 0.28$, a very low cutoff latency that includes all settings in which a context switch is at least as time-consuming as an individual action (i.e. $L \geq 1$).

In experiments, we find that SafeDAgger-Execution's short interventions lead to many instances of Failure Mode A (see Table 2.1), as the policy is making task progress, but not quickly enough to perform the task. We observe that Failure Mode C is often due to the fabric

reaching a highly irregular configuration that is not within the training data distribution, making it difficult for the robot policy to make progress. We find that SafeDAgger and LazyDAgger experience Failure Mode D at a similar rate as they use the same criteria to solicit interventions (but different termination criteria). However, we find that all of LazyDAgger's failures are due to Failure Mode D, while SafeDAgger also fails in Mode A due to premature termination of interventions.

2.7 Discussion and Future Work

We propose context switching between robot and human control as a metric for supervisor burden in interactive imitation learning and present LazyDAgger, an algorithm which can be used to efficiently learn tasks while reducing this switching. We evaluate LazyDAgger on 3 continuous control benchmark environments in MuJoCo, a fabric smoothing environment in simulation, and a fabric manipulation task with an ABB YuMi robot and find that LazyDAgger is able to improve task performance while reducing context switching between the learner and robot by up to 79% over SafeDAgger. In the next chapter, we investigate improved intervention criteria and apply robot-gated interventions to controlling a small fleet of robots, where context switching can negatively impact task throughput.

Chapter 3

ThriftyDAgger: Budget-Aware Novelty and Risk

In this chapter, we introduce ThriftyDAgger, another novel algorithm for robot-gated interactive imitation learning. ThriftyDAgger addresses shortcomings in LazyDAgger and uses novelty and risk estimation to significantly outperform prior work in balancing task performance with burden on the human supervisor.

3.1 Introduction

Imitation learning (IL) [12, 15, 187] has seen success in a variety of robotic tasks ranging from autonomous driving [203, 191, 53] to robotic manipulation [74, 73, 80, 244, 133]. In its simplest form, the human provides an offline set of task demonstrations to the robot, which the robot uses to match human behavior. However, this offline approach can lead to low task performance due to a mismatch between the state distribution encountered in the demonstrations and that visited by the robot [215, 138], resulting in brittle policies that cannot be effectively deployed in real-world applications [108]. *Interactive imitation learning*, in which the robot periodically cedes control to a human supervisor for corrective interventions, has emerged as a promising technique to address these challenges [125, 241, 110, 98]. However, while interventions make it possible to learn robust policies, this can come at the expense of requiring the human to spend significant time providing these interventions. Thus, the central challenge in interactive IL algorithms is to control the timing and length of interventions to strike a balance between task performance and the burden imposed on the human supervisor [276, 98]. Achieving such a balance is even more critical if the human supervisor must oversee multiple robots at once [58, 38, 245], for instance supervising a robot fleet in a warehouse or manufacturing setting.

One way to determine when to solicit interventions is to allow the human supervisor to decide when to provide the corrective interventions. However, these approaches—termed "human-gated" interactive IL algorithms [125, 241, 160]—require the human supervisor to

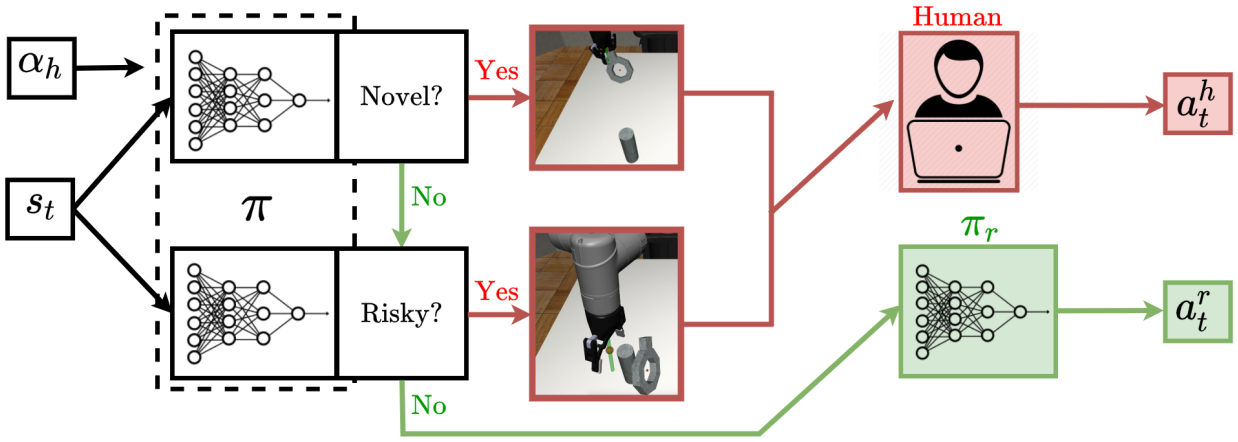


Figure 3.1: **ThriftyDAgger:** Given a desired context switching rate α_h, ThriftyDAgger transfers control to a human supervisor if the current state s_t is (1) sufficiently novel or (2) sufficiently risky, indicating that the probability of task success is low under robot policy π_r. Intuitively, one should not only distrust π_r in states significantly out of the distribution of previously-encountered states, but should also cede control to a human supervisor in more familiar states where the robot predicts that it is unlikely to successfully complete the task.

continuously monitor the robot to determine when to intervene. This imposes significant burden on the supervisor and cannot effectively scale to settings in which a small number humans supervise a large number of robots. To address this challenge, there has been recent interest in approaches that enable the robot to actively query humans for interventions, called "robot-gated" algorithms [276, 139, 171, 98]. Robot-gated methods allow the robot to reduce burden on the human supervisor by only requesting interventions when necessary, switching between robot control and human control based on some intervention criterion. Hoque et al. [98] formalize the idea of supervisor burden as the expected total cost incurred by the human in providing interventions, which consists of the expected cost due to *context switching* between autonomous and human control and the time spent actually providing interventions. However, it is difficult to design intervention criteria that limit this burden while ensuring that the robot gains sufficient information to imitate the supervisor's policy.

This paper makes several contributions. First, we develop intervention criteria based on a synthesis of two estimated properties of a given state: *novelty*, which measures whether the state is significantly out of the distribution of previously encountered states, indicating that the robot policy should not be trusted; and *risk*, which indicates whether the robot is unlikely to make task progress. In particular, we propose a novel risk metric estimating the probability of task success. Second, we present a novel robot-gated interactive IL algorithm, ThriftyDAgger (Figure 3.1), which employs these measures jointly to solicit human interventions only when necessary. Third, while prior robot-gated algorithms [276, 98] require careful parameter tuning to modulate the timing and frequency of human intervention requests, ThriftyDAgger only requires the supervisor to specify a desired context

switching rate and automatically tunes other parameters accordingly. Fourth, experimental results demonstrate ThriftyDAgger's effectiveness for reducing supervisor burden while learning challenging tasks both in simulation and in an image-based cable routing task on a physical robot. Finally, the results of a human user study applying ThriftyDAgger to control a fleet of three simulated robots suggest that ThriftyDAgger significantly improves performance on both the robots' task and an independent human task while imposing fewer context switches, fewer human intervention actions, and lower mental load and frustration than prior algorithms.

3.2 Related Work

Imitation Learning from Human Feedback: There has been significant prior work in offline imitation learning, in which the agent leverages an offline dataset of expert demonstrations either to directly match the distribution of trajectories in the offline dataset [203, 95, 12, 187, 13, 106, 194], for instance via Behavior Cloning [253, 54], or to learn a reward function that can then be optimized via reinforcement learning [1, 95, 29]. However, while these approaches have shown significant success in a number of domains [74, 244, 80, 54], learning from purely offline data leads to a mismatch in the state distributions of the trajectories used for learning and those visited by the agent's policy, leading to suboptimal performance both in theory and practice [215, 138]. To address this problem, there have been a number of approaches that utilize online human feedback while the agent acts in the environment, such as providing suggested actions [215, 16, 115, 110] or preferences [220, 49, 102, 190, 21, 30]. However, many of these forms of human feedback may be unreliable if the robot visits states that significantly differ from those the human supervisor would themselves visit; in such situations, it is challenging for the supervisor to determine what correct behavior should look like without directly interacting with the environment [241, 208]. Furthermore, allowing only the robot to act in the environment can be unsafe, and most of these algorithms result in relatively high supervisor burden. DAgger [215] requires the supervisor to suggest an action at every timestep, and the preference learning approaches require the supervisor to indicate a preference over every pair of executed actions or trajectories. SHIV [139] reduces the required number of suggested actions but still faces the aforementioned pitfalls.

Interactive Imitation Learning: A natural way to collect reliable online feedback for imitation learning is to periodically cede control to a human supervisor, who then provides a corrective intervention to illustrate desired behavior. Human-gated interactive IL algorithms [125, 241, 160] such as HG-DAgger require the human to determine when to engage in interventions. However, these algorithms require a human to continuously monitor the robot to determine when to intervene, which imposes significant burden on the supervisor and is particularly impractical if a small number of humans must supervise a large number of robots or if a human must supervise for a very long period of time. Furthermore, it requires the human to determine when the robot needs help and when to cede control, which can be unintuitive and result in suboptimal switching points.

By contrast, robot-gated interactive IL algorithms, such as EnsembleDAgger [171], Safe DAgger [276], and LazyDAgger [98], allow the robot to solicit interventions from a human when the system deems necessary. In practice, these algorithms estimate various quantities correlated with task performance [276, 98, 214, 139] and uncertainty [171] and use them to determine when to solicit interventions. Prior work has proposed intervention criteria which use the novelty of states visited by the robot [171] or the predicted discrepancy between the actions proposed by the robot policy and by the supervisor [276, 98]. However, while state novelty provides a valuable signal for soliciting interventions, we argue that this alone is insufficient, as a state's novelty does not convey information about the level of precision with which actions must be executed in that state. In practice, many robotic tasks involve moving through critical "bottlenecks" [160], which, though not necessarily novel, still present challenges. Examples include moving an eating utensil close to a person's mouth or placing an object on a shelf without disturbing nearby objects. Similarly, even if predicted accurately, action discrepancy is often a flawed risk measure, as high action discrepancy between the robot and the supervisor may be permissible when fine-grained control is not necessary (e.g. a robot gripper moving in free space) but impermissible when precision is critical (e.g. a robot gripper actively trying to grasp an object). In contrast, ThriftyDAgger carefully designs asymmetric intervention criteria incorporating both state novelty and a novel risk metric, allowing more efficient use of human supervision. ThriftyDAgger also significantly reduces the need for parameter tuning, to which prior algorithms can be highly sensitive [171].

3.3 Problem Statement

Given a robot, a task for the robot to accomplish, and a human supervisor with a specified context switching budget, the goal is to train the robot to imitate supervisor performance within the budget. We model the robot environment as a discrete-time Markov Decision Process (MDP) $\mathcal{M}$ with continuous states $s \in \mathcal{S}$, continuous actions $a \in \mathcal{A}$, and time horizon T [205]. We consider the interactive imitation learning (IL) setting [125], where the robot does not have access to a shaped reward function or transition dynamics of the MDP but can temporarily cede control to a supervisor who uses policy $\pi_H : \mathcal{S} \to \mathcal{A}$. We specifically focus on tasks where there is a goal set $\mathcal{G}$ which determines success, but that can be challenging and long-horizon, making direct application of RL highly sample inefficient.

We assume that the human and robot utilize the same action space (e.g. through a teleoperation interface) and that task success can be specified by convergence to some goal set $\mathcal{G} \subseteq \mathcal{S}$ within the time horizon (i.e., the task is successful if $\mathcal{G}$ is reached within T timesteps). We further assume access to an indicator function $\mathbb{1}_{\mathcal{G}} : \mathcal{S} \to \{0, 1\}$, which indicates whether a state belongs to the goal set $\mathcal{G}$.

The IL objective is to minimize a surrogate loss function $J(\pi_R)$ to encourage the robot

policy $\pi_R : \mathcal{S} \to \mathcal{A}$ to match π_H:

$$J(\pi_R) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi_R}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right], \tag{3.1}$$

where $\mathcal{L}(\pi_R(s), \pi_H(s))$ is an action discrepancy measure between $\pi_R(s)$ and $\pi_H(s)$ (e.g. MSE loss), and $d_t^{\pi_R}$ is the marginal state distribution at timestep t induced by the robot policy π_R in $\mathcal{M}$.

In the interactive IL setting, meanwhile, in addition to optimizing Equation 3.1, a key design goal is to minimize the imposed burden on the human supervisor. To formalize this, we define a switching policy π, which determines whether the system is under robot control π_R (which we call *autonomous mode*) or human supervisor control π_H (which we call *supervisor mode*). Following prior work [98], we define $C(\pi)$, the expected number of *context switches* in an episode under policy π, as follows: $C(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi}} [m_I(s_t; \pi)]$, where $m_I(s_t; \pi)$ is an indicator for whether or not a context switch occurs from autonomous to supervisor control. Similarly, we define $I(\pi)$ as the expected number of *supervisor actions* in an intervention solicited by π. We then define the total burden $B(\pi)$ imposed on the human supervisor as follows:

$$B(\pi) = C(\pi) \cdot \big(L + I(\pi)\big), \tag{3.2}$$

where L is the *latency* of a context switch between control modes (summed over both switching directions) in units of timesteps (one action per timestep). The interactive IL objective is to minimize the discrepancy from the supervisor policy while limiting supervisor burden within some Γ_{b}:

$$\pi = \arg\min_{\pi' \in \Pi} \{ J(\pi_R) \mid B(\pi') \leq \Gamma_{\mathrm{b}} \}. \tag{3.3}$$

3.4 ThriftyDAgger

ThriftyDAgger determines when to switch between autonomous and human supervisor control modes by leveraging estimates of both the *novelty* and *risk* of states. Below, we discuss the estimation of state novelty and risk of task failure, ThriftyDAgger's integration of these measures to determine when to switch control modes, a procedure to automatically tune key parameters to regulate switches between control modes, and the full control flow of ThriftyDAgger.

Novelty Estimation

When the robot policy visits states that lie significantly outside the distribution of those encountered in the supervisor trajectories, it does not have any reference behavior to imitate. This motivates initiating interventions to illustrate desired recovery behaviors in these states.

However, estimating the support of the state distribution visited by the human supervisor is challenging in the high-dimensional state spaces common in robotics. Following prior work [171], we train an ensemble of policies with bootstrapped samples of transitions from supervisor trajectories. We then measure the novelty of a given state s by calculating the variance of the policy outputs at state s across ensemble members. In practice, the action $a \in \mathcal{A}$ outputted by each policy is a vector; thus, we measure state novelty by computing the variance of each component of the action vector a across the ensemble members and then averaging over the components. We denote this quantity by Novelty(s). Once in supervisor mode, as noted in Hoque et al. [98], we can obtain a more precise correlate of novelty by computing the ground truth action discrepancy between actions suggested by the supervisor and the robot policy.

Risk Estimation

Interventions may be required not only in novel states outside the distribution of supervisor trajectories, but also in familiar states that are prone to result in task failure. To address this challenge, we propose a novel measure of a state's "riskiness," capturing the likelihood that the robot cannot successfully converge to the goal set $\mathcal{G}$. We first define a Q-function to quantify the discounted probability of successful convergence to $\mathcal{G}$ from a given state and action under the robot policy:

$$Q_{\mathcal{G}}^{\pi_r}(s_t, a_t) = \mathbb{E}_{\pi_r}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} \mathbb{1}_{\mathcal{G}}(s'_t) | s_t, a_t\right], \tag{3.4}$$

where $\mathbb{1}_{\mathcal{G}}(s_t)$ is equal to 1 if s_t belongs to $\mathcal{G}$. We estimate $Q_{\mathcal{G}}^{\pi_r}(s_t, a_t)$ via a function approximator $\hat{Q}_{\phi,\mathcal{G}}^{\pi_r}$ parameterized by ϕ, and define a state's riskiness in terms of this learned Q-function:

$$\text{Risk}^{\pi_r}(s, a) = 1 - \hat{Q}_{\phi,\mathcal{G}}^{\pi_r}(s, a). \tag{3.5}$$

In practice, we train $\hat{Q}_{\phi,\mathcal{G}}^{\pi_r}$ on transitions (s_t, a_t, s_{t+1}) collected by the supervisor from both offline data and online interventions by minimizing the following MSE loss inspired by [247]:

$$J_{\mathcal{G}}^{Q}(s_t, a_t, s_{t+1}; \phi) = \frac{1}{2}\left(\hat{Q}_{\phi,\mathcal{G}}^{\pi_r}(s_t, a_t) - (\mathbb{1}_{\mathcal{G}}(s_t) + (1 - \mathbb{1}_{\mathcal{G}}(s_t))\gamma \hat{Q}_{\phi,\mathcal{G}}^{\pi_r}(s_{t+1}, \pi_r(s_{t+1})))\right)^2. \tag{3.6}$$

Note that since $\hat{Q}_{\phi,\mathcal{G}}^{\pi_r}$ is only used to solicit interventions, it need only be accurate enough to distinguish risky states from others, rather than be able to make the fine-grained distinctions between different states required for accurate policy learning in reinforcement learning.

Regulating Switches in Control Modes

We now describe how ThriftyDAgger leverages the novelty estimator from Section 3.4 and the risk estimator from Section 3.4 to regulate switches between autonomous and supervisor

control. While in autonomous mode, the switching policy π initiates a switch to supervisor mode at timestep t if state s_t is either (1) sufficiently unfamiliar or (2) the robot policy has a low probability of task success from s_t. Stated precisely, π initiates a switch to supervisor mode from autonomous mode at timestep t if the predicate $\text{Intervene}(s_t, \delta_h, \beta_H)$ evaluates to TRUE, where $\text{Intervene}(s_t, \delta_h, \beta_H)$ is TRUE if (1) $\text{Novelty}(s_t) > \delta_h$ or (2) $\text{Risk}^{\pi_r}(s_t, \pi_r(s_t)) > \beta_H$ and FALSE otherwise. Note that the proposed switching policy only depends on Risk^{π_r} for states which are *not* novel (as novel states already initiate switches to supervisor control regardless of risk), since the learned risk measure should only be trusted on states in the neighborhood of those on which it has been trained.

In supervisor mode, π switches to autonomous mode if the action discrepancy between the human and robot policy and the robot's task failure risk are both below threshold values (Section 3.4), indicating that the robot is in a familiar and safe region. Stated precisely, π switches to autonomous mode from supervisor mode if the predicate $\text{Cede}(s_t, \delta_r, \beta_R)$ evaluates to TRUE, where $\text{Cede}(s_t, \delta_r, \beta_R)$ is TRUE if (1) $||\pi_R(s_t) - \pi_H(s_t)||_2^2 < \delta_r$ and (2) $\text{Risk}^{\pi_r}(s_t, \pi_r(s_t)) < \beta_R$, and FALSE otherwise. While the former condition prevents reliance on the learned risk measure in unfamiliar regions, the latter prevents switching to autonomous mode when precise control may be necessary. Motivated by prior work [98] and hysteresis control [23], we use stricter switching criteria in supervisor mode ($\beta_R < \beta_H$) to encourage longer interventions and reduce context switches experienced by the human supervisor.

Automatic Parameter Tuning

One challenge of the control strategy presented in Section 3.4 lies in tuning the key parameters $(\delta_h, \delta_r, \beta_H, \beta_R)$ governing when context switching occurs. As noted in prior work [171], performance and supervisor burden can be sensitive to these thresholds. To address this difficulty, we assume that the user specifies their availability in the form of a desired intervention budget $\alpha_h \in [0, 1]$, indicating the desired proportion of timesteps in which interventions will be requested. This desired context switching rate can be interpreted in the context of supervisor burden as defined in Equation (3.2): if the latency of a context switch dominates the time cost of the intervention itself, limiting the expected number of context switches to within some intervention budget directly limits supervisor burden.

Given α_h, we set β_H to be the $(1-\alpha_h)$-quantile of $\text{Risk}^{\pi_r}(s, \pi_r(s))$ for all states previously visited by π_R and set δ_h to be the $(1-\alpha_h)$-quantile of $\text{Novelty}(s)$ for all states previously visited by π_R. We set δ_r to be the mean action discrepancy on the states visited by the supervisor after π_R is trained and set β_R to be the median of $\text{Risk}^{\pi_r}(s, \pi_r(s))$ for all states previously visited by π_R. (Note that β_R can easily be set to different quantiles to adjust mean intervention length if desired.) We find that these settings strike a balance between informative interventions and imposed supervisor burden.

ThriftyDAgger Overview

We now summarize the ThriftyDAgger procedure, with full pseudocode available in the supplement. ThriftyDAgger first initializes π_r via Behavior Cloning on offline transitions ($\mathcal{D}_h$ from the human supervisor, π_h). Then, π_r collects an initial offline dataset $\mathcal{D}_r$ from the resulting π_r, initializes $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ by optimizing Equation (3.5) on $\mathcal{D}_r \cup \mathcal{D}_h$, and initializes parameters $\beta_H, \beta_R, \delta_h$, and δ_r as in Section 3.4. We then collect data for N episodes, each with up to T timesteps. In each timestep of each episode, we determine whether robot policy π_r or human supervisor π_h should be in control using the procedure in Section 3.4. Transitions in autonomous mode are aggregated into $\mathcal{D}_r$ while transitions in supervisor mode are aggregated into $\mathcal{D}_h$. After each episode, π_r is updated via supervised learning on $\mathcal{D}_h$, while $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ is updated on $\mathcal{D}_r \cup \mathcal{D}_h$ to reflect the probability of task success of the resulting π_r.

3.5 Experiments

In the following experiments, we study whether ThriftyDAgger can balance task performance and supervisor burden more effectively than prior IL algorithms in three contexts: (1) training a simulated robot to perform a peg insertion task; (2) supervising a fleet of three simulated robots to perform the peg insertion task in a human user study; and (3) training a physical surgical robot to perform a cable routing task.

Evaluation Metrics

We consider ThriftyDAgger's performance during training and execution. For the latter, we evaluate both the (1) *autonomous success rate*, or success rate when deployed after training without access to a human supervisor, and (2) *intervention-aided success rate*, or success rate when deployed after training with a human supervisor in the loop. These metrics are reported in the Peg Insertion study (Section 3.5) and the Physical Cable Routing study (Section 3.5). For all experiments, during both training and intervention-aided execution, we evaluate the number of interventions per episode and the numbers of human and robot actions per episode. During execution time, all three of these metrics are computed using successful episodes only, in order not to bias the metrics by the maximum episode horizon length, T. In our user study (Section 3.5), we also report the following quantities: throughput (total number of task successes across the three robots), performance on an independent human task, the idle time of the robots in the fleet, and users' qualitative ratings of mental load and frustration.

Comparisons

We compare ThriftyDAgger to the following algorithms: Behavior Cloning, which does not use interventions; HG-DAgger [125], which is human-gated and always requires supervision;

SafeDAgger [276], which is robot-gated and performs interventions based on estimated action discrepancy between the human supervisor and robot policy; and LazyDAgger [98], which builds on SafeDAgger by introducing an asymmetric switching criterion to encourage lengthier interventions. We also implement two ablations: one that does not use a novelty measure to regulate context switches (ThriftyDAgger (-Novelty)) and one that does not use risk to regulate context switches (ThriftyDAgger (-Risk)).

Peg Insertion in Simulation

We first evaluate ThriftyDAgger on a long-horizon peg insertion task (Figure 3.2) from the Robosuite simulation environment [280]. The goal is to grasp a ring in a random initial pose and thread it over a cylinder at a fixed target location. This task has two bottlenecks which motivate learning from interventions: (1) correctly grasping the ring and (2) correctly placing it over the cylinder (Figure 3.2). A human teleoperates the robot through a keyboard interface to provide interventions. The states consist of the robot's joint angles and ring's pose, while the actions specify 3D translation, 3D rotation, and opening or closing the gripper. For ThriftyDAgger and its ablations, we use target intervention frequency $\alpha_h = 0.01$ and set other parameters via the automated tuning method (Section 3.4). We collect 30 offline task demos (2,687 state-action pairs) from a human supervisor to initialize the robot policy for all compared algorithms. Behavior Cloning is given 15 additional offline demos for a fair comparison. For ThriftyDAgger and each interactive IL baseline, we perform 10,000 environment steps, during which each episode takes at most 175 timesteps and system control switches between the human and robot. Hyperparameter settings for all algorithms are detailed in the supplement.

Results (Table 3.1) suggest that ThriftyDAgger achieves a significantly higher autonomous success rate than prior robot-gated algorithms, although it does request more human actions due to its conservative exit criterion for interventions (Cede(s_t, δ_r, β_R)). However, the number of interventions is similar to prior robot-gated algorithms, indicating that while ThriftyDAgger requires more human actions, it imposes a similar supervisor burden to SafeDAgger and LazyDAgger in settings in which context switches are expensive or time-consuming (e.g. high latency L in Equation 3.2). We find that all interactive IL algorithms substantially outperform Behavior Cloning, which does not have access to supervisor interventions. Notably, ThriftyDAgger achieves a higher autonomous success rate than even HG-DAgger, in which the supervisor is able to decide the timing and length of interventions. This indicates that ThriftyDAgger's intervention criteria enable it to solicit more informative interventions than those chosen by a human supervisor. Furthermore, ThriftyDAgger achieves a 100% intervention-aided success rate at execution time, suggesting that ThriftyDAgger successfully identifies the required states at which to solicit interventions. We find that both ablations of ThriftyDAgger (Ours (-Novelty) and Ours (-Risk)) achieve significantly lower autonomous success rates, indicating that both the novelty and risk measures are critical to ThriftyDAgger's performance. We calculate ThriftyDAgger's context switching rate to be

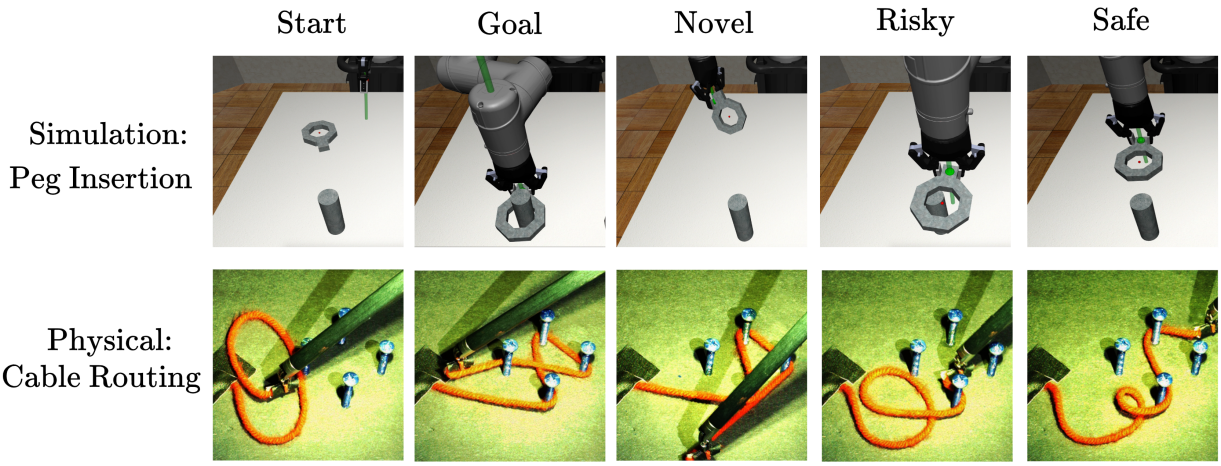


Figure 3.2: **Experimental Domains:** We visualize the peg insertion simulation domain (top row) and the physical cable routing domain with the physical robot (bottom row). We visualize sample start and goal states, in addition to states which ThriftyDAgger categorizes as novel, risky, and neither. ThriftyDAgger marks states as novel if they are far from behavior that the supervisor would produce, and risky if it is stuck in a bottleneck, e.g. if the ring is wedged against the side of the cylinder (top) or the cable is near all four obstacles (bottom).

1.15% novelty switches and 0.79% risk switches, both approximately in the range specified by the budget of $\alpha_h = 0.01$.

User Study: Controlling A Fleet of Three Robots in Simulation

We conduct a user study with 10 participants (7 male and 3 female, aged 18-37). Participants supervise a fleet of three simulated robots, each performing the peg insertion task from Section 3.5. We evaluate how different interactive IL algorithms affect the participants' (1) ability to provide effective robot interventions, (2) performance on a distractor task performed between robot interventions, and (3) levels of mental demand and frustration. For the distractor task, we use the game Concentration (also known as Memory or Matching Pairs), in which participants identify as many pairs of matching cards as possible among a set of face-down cards. This is intended to emulate tasks which require continual focus, such as cooking a meal or writing a research paper, in which frequent context switches between performing the task and helping the robots is frustrating and degrades performance.

The participants teleoperate the robots using three robot-gated interactive IL algorithms: SafeDAgger, LazyDAgger, and ThriftyDAgger. The participant is instructed to make progress on the distractor task only when no robot requests an intervention. When an intervention is requested, the participant is instructed to pause the distractor task, provide an intervention from the requested state until the robot (or multiple robots queued after each other) no longer requires assistance, and then return to the distractor task. The participants also teleoperate with HG-DAgger, where they no longer perform the distractor task and are

Algorithm	Training Interventions			Auto Succ.	Execution Interventions			Int-Aided Succ.
	Ints	Acts (H)	Acts (R)		Ints	Acts (H)	Acts (R)	
Behavior Cloning	N/A	N/A	108.0 ± 15.9	24/100	N/A	N/A	N/A	N/A
SafeDAgger	3.89 ± 1.44	19.8 ± 9.9	88.8 ± 19.4	24/100	4.00 ± 1.37	19.5 ± 5.3	77.5 ± 11.7	17/20
LazyDAgger	1.46 ± 1.15	13.2 ± 12.4	102.1 ± 18.2	48/100	1.73 ± 1.29	12.6 ± 14.4	91.7 ± 24.0	11/20
HG-DAgger	1.49 ± 0.88	20.3 ± 15.6	97.1 ± 17.5	57/100	1.15 ± 0.73	17.1 ± 11.6	103.6 ± 14.0	**20/20**
Ours (-Novelty)	$\mathbf{0.79 \pm 0.81}$	35.1 ± 23.1	70.0 ± 35.8	49/100	$\mathbf{0.33 \pm 0.62}$	2.5 ± 5.0	114.0 ± 26.0	12/20
Ours (-Risk)	0.99 ± 0.96	7.8 ± 12.0	104.2 ± 19.2	49/100	1.39 ± 0.95	9.8 ± 12.0	109.1 ± 22.9	18/20
Ours: ThriftyDAgger	0.88 ± 1.01	43.6 ± 24.5	60.0 ± 32.8	**73/100**	1.35 ± 0.66	21.3 ± 15.0	84.8 ± 21.8	**20/20**

Table 3.1: **Peg Insertion in Simulation Results:** We first report training performance (number of interventions (Ints), number of human actions (Acts (H)), and number of robot actions (Acts (R))) and report the success rate of the fully-trained policy at execution time when no interventions are allowed (Auto Succ.). We then evaluate the fully-trained policies with interventions allowed and report the same intervention statistics and the success rate (Int-Aided Succ.). We find that ThriftyDAgger achieves the highest autonomous and intervention-aided success rates among all algorithms compared. Notably, ThriftyDAgger even achieves a higher autonomous success rate than HG-DAgger, in which the human decides when to intervene during training.

instructed to continually monitor all three robots simultaneously and decide on the length and timing of interventions themselves. Each algorithm runs for 350 timesteps, where in each timestep, all robots in autonomous mode execute one action and the human executes one action on the currently-supervised robot (if applicable). The supplement illustrates the user study interface and fully details the experiment protocol. All algorithms are initialized as in Section 3.5.

Results (Table 3.2) suggest that ThriftyDAgger achieves significantly higher throughput than all prior algorithms while requiring fewer interventions and fewer human actions, indicating that ThriftyDAgger requests interventions more judiciously than prior algorithms. Furthermore, ThriftyDAgger also enables a lower mean idle time for robots and higher performance on the distractor task. Notably, ThriftyDAgger solicits fewer interventions and total actions while achieving a higher throughput than even HG-DAgger, in which the participant chooses when to intervene. We also report metrics of users' mental workload and frustration using the NASA-TLX scale [93] in the appendix. Results suggest that users experience lower degrees of frustration and mental load when interacting with ThriftyDAgger and LazyDAgger compared to HG-DAgger and SafeDAgger. We hypothesize that participants struggle with HG-DAgger due to the difficultly of monitoring multiple robots simultaneously, while SafeDAgger's frequent context switches lead to user frustration during experiments.

Physical Experiment: Visuomotor Cable Routing

Finally, we evaluate ThriftyDAgger on a long-horizon cable routing task with a da Vinci surgical robot [119]. Here, the objective is to route a red cable into a Figure-8 pattern around 4 pegs via teleoperation with the robot's master controllers (see supplement). The algorithm

Algorithm	Interventions	Human Actions	Robot Actions	Concentration Pairs	Throughput	Mean Idle Time
HG-DAgger	10.6 ± 2.5	198.0 ± 32.1	834.4 ± 38.1	N/A	5.1 ± 1.9	N/A
SafeDAgger	22.1 ± 4.8	234.1 ± 31.8	700.7 ± 70.4	17.7 ± 8.2	3.0 ± 2.4	38.4 ± 14.1
LazyDAgger	10.0 ± 2.1	219.5 ± 43.3	719.2 ± 89.7	20.9 ± 7.9	5.1 ± 1.7	37.1 ± 20.5
Ours: ThriftyDAgger	**7.9 ± 2.1**	**179.4 ± 34.9**	793.2 ± 86.6	**33.0 ± 8.5**	**9.2 ± 2.0**	**25.8 ± 19.3**

Table 3.2: **Three-Robot Fleet Control User Study Results:** Results for experiments with 10 human subjects and 3 simulated robots on the peg insertion task. We report the total numbers of interventions, human actions, and robot actions, as well as the throughput, or total task successes achieved across robots, for all algorithms. Additionally, for robot-gated algorithms, we report the Concentration score (number of pairs found) and the mean idle time of robots in the fleet in timesteps. Results suggest that ThriftyDAgger outperforms all prior algorithms across all metrics, requesting fewer interventions and total human actions while achieving higher throughput, lowering the robots' mean idle time, and enabling higher performance on the Concentration task.

only observes high-dimensional $64 \times 64 \times 3$ RGB images of the workspace and generates continuous actions representing delta-positions in (x, y). As in Section 3.5, ThriftyDAgger uses a target intervention frequency of $\alpha_h = 0.01$. We collect 25 offline task demonstrations (1,381 state-action pairs) from a human supervisor to initialize the robot policy for ThriftyDAgger and all comparisons. We perform 1,500 environment steps, where each episode has at most 100 timesteps and system control can switch between the human and robot. The supplement details the hyperparameter settings for all algorithms.

Results (Table 3.3) suggest that both ThriftyDAgger and HG-DAgger achieve a significantly higher autonomous success rate than Behavior Cloning, which is never able to complete the task. Furthermore, ThriftyDAgger achieves a higher autonomous success rate than even HG-DAgger while requesting fewer interventions and a similar number of total human actions. This again suggests that ThriftyDAgger's intervention criteria enable it to solicit more informative interventions than those chosen by a human supervisor. Finally, at execution time ThriftyDAgger achieves a 100% intervention-aided success rate with minimal supervision, again indicating that ThriftyDAgger successfully identifies the required states at which solicit interventions to increase policy reliability.

3.6 Discussion and Future Work

We present ThriftyDAgger, a scalable robot-gated interactive imitation learning algorithm that leverages learned estimates of state novelty and risk of task failure to reduce burden on a human supervisor during training and execution. Experiments suggest that ThriftyDAgger effectively enables long-horizon robotic manipulation tasks in simulation, on a physical robot, and for a three-robot fleet while limiting burden on a human supervisor.

One direction for future work is casting ThriftyDAgger in the context of goal-conditioned reinforcement learning, where the learned risk measure is leveraged to update the robot policy

Algorithm	Training Interventions			Auto Succ.	Execution Interventions			Int-Aided Succ.
	Ints	Acts (H)	Acts (R)		Ints	Acts (H)	Acts (R)	
Behavior Cloning	N/A	N/A	N/A	0/15	N/A	N/A	N/A	N/A
HG-DAgger	1.55 ± 1.16	13.9 ± 10.9	55.5 ± 10.9	10/15	$\mathbf{0.40 \pm 0.49}$	2.7 ± 3.5	73.9 ± 7.9	**15/15**
Ours: ThriftyDAgger	$\mathbf{1.42 \pm 1.14}$	15.2 ± 12.4	45.5 ± 18.3	**12/15**	0.40 ± 0.71	1.5 ± 3.1	61.3 ± 6.5	**15/15**

Table 3.3: **Physical Cable Routing Results:** We first report intervention statistics during training (number of interventions (Ints), number of human actions (Acts (H)), and number of robot actions (Acts (R))) and report the success rate of the fully-trained policy at execution time when no interventions are allowed (Auto Succ.). We then evaluate the fully-trained policies with interventions allowed and report the same intervention statistics and the success rate (Int-Aided Succ.). We find that ThriftyDAgger achieves the highest autonomous and intervention-aided success rates among all algorithms compared. Notably, ThriftyDAgger even achieves a higher autonomous success rate than HG-DAgger, in which the human decides when to intervene during training.

to potentially outperform the demonstrator by explicitly minimizing the likelihood of task failure. In addition, it would interesting to experimentally evaluate how ThriftyDAgger's performance varies with the target supervisor burden (specified via α_h). In practice, α_h could be time-varying: for instance, α_h may be significantly lower at night, when human operators may have limited availability. Similarly, α_h may be set to a higher value during training than at deployment, when the robot policy is typically higher quality.

In Chapter 5, we evaluate ThriftyDAgger and other algorithms in fleet learning experiments with a large number of simulated and physical robots learning different tasks.

Chapter 4

IntervenGen: Interventional Data Generation

In this chapter, we present IntervenGen, an alternative approach to facilitating scalability in interactive IL. From a small set of 10 human interventions, IntervenGen can autonomously generate thousands of new interventions.

4.1 Introduction

Imitation learning (IL) from human demonstrations is a promising paradigm for training robot policies. One approach is to collect a set of offline task demonstrations via human teleoperation [163, 158] and employ behavior cloning (BC) [204] to train robot policies via supervised learning, where the labels are robot actions. There have been recent efforts to scale this approach by collecting thousands of demonstrations using hundreds of human operator hours and training high-capacity neural networks on the large-scale data [109, 27, 69, 2, 152].

However, IL policies can suffer from distribution shift, where the conditions at evaluation time differ from those in the training data [215]. As an example, consider a policy that makes decisions based on object pose observations. A common source of distribution shift in the real world is object pose estimation error, which can occur due to a wide range of factors such as sensor noise, occlusion, network delay, and model misspecification. This can cause inaccuracy in the robot's belief of where critical objects are located in the environment, leading the robot to visit states outside the training distribution that result in poor policy performance.

One approach to addressing distribution shift is to collect a large set of demonstrations under diverse conditions and hope that agents trained on this data can generalize. However, human teleoperation data is notoriously difficult to collect due to the human time, effort, and financial cost required [109, 27, 69, 2, 152].

An alternative approach is interactive IL (i.e., DAgger [215] and variants [125, 160, 97]),

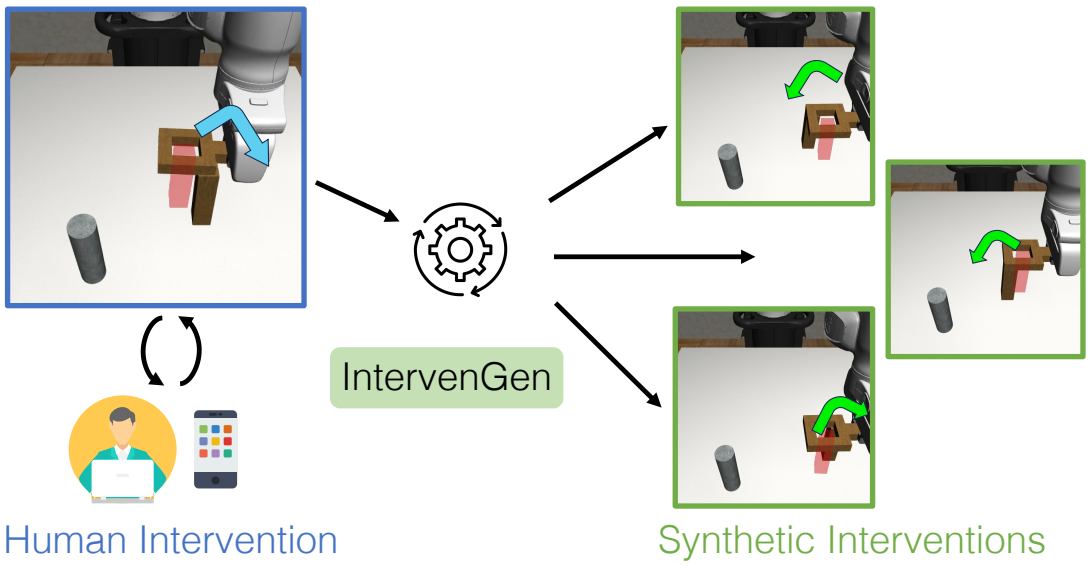


Figure 4.1: **Overview.** IntervenGen automatically generates corrective interventional data from a small number of human interventions, with coverage across both diverse scene configurations and policy mistake distributions. Here, the robot mistakenly believes the peg is at the position highlighted in red and requires demonstration of recovery behavior toward the true peg position.

where humans can intervene during robot execution and demonstrate *recovery behaviors* to help the robot return to the support of the training distribution. Subsequent training on these corrections can increase policy robustness and performance both theoretically and in practice [215]. However, interactive IL imposes even more burden on the human supervisors than behavior cloning, as the human must continuously monitor robot task execution and intervene when they see fit, typically over multiple rounds of interleaved data collection and policy training. Moreover, a significant amount of recovery data may be required to adequately cover the distribution of mistakes the policy may make.

We raise the following question: do we actually need to have a human operator collect corrections every single time a policy makes a mistake? MimicGen [159], a recently proposed data generation system, raises an intriguing possibility: a large dataset of synthetically generated demonstrations derived from a small set of human demonstrations (typically 100× smaller or more) can produce performant robot policies. The system's key insight is that similar object-centric manipulation behaviors can be applied in new contexts by appropriately transforming demonstrated behavior to the new object frame. Inspired by this insight, we propose a data generation system for *interventional* data (see Fig. 4.1). With a small set of corrective interventions from a human operator, we can autonomously generate data with significantly higher coverage of the distribution of potential policy mistakes. Our system can be applied to a broad range of applications such as improving policy success rates on a

task of interest, making policies robust to errors in perception, and more broadly, acting as a domain randomization [250] procedure to aid in sim-to-real transfer of IL policies without requiring additional data collection from a human supervisor. In this work, we focus on improving policy robustness to errors in perception.

We make the following contributions:

1. IntervenGen (I-Gen), a system for automatically generating interventional data across diverse scene configurations and broad mistake distributions from a small number of human interventions.
2. An application of I-Gen to improve policy robustness against 2 sources of object pose estimation error (sensor noise and geometry error) in 5 high-precision 6-DOF manipulation tasks. I-Gen increases policy robustness by up to 39× with only 10 human interventions.
3. Experiments demonstrating the utility of I-Gen over alternate uses of a human data budget of equivalent or even greater size. A policy trained on synthetic I-Gen data from 10 source human interventions can outperform one trained on even 100 human interventions by 24%, with 12% of the data collection time and effort.
4. An experiment that shows that policies trained in simulation with I-Gen are amenable to real-world deployment and retain robustness to erroneous state estimation.

4.2 Related Work

Data Collection Approaches for Robot Learning. Many prior works address the need for large-scale data in robotics. Some use self-supervised data collection [118, 64], but the data can have low signal-to-noise ratio due to the trial-and-error process. Other works collect large datasets using experts that operate on privileged information available in simulation [113, 63, 169]. Still, designing such experts can require significant engineering. One popular approach is to collect demonstrations by having human operators teleoperate robot arms [163, 158, 27, 109]; however, this can require hundreds of hours of human operator time. Some systems also allow for collecting interventions to help correct policy mistakes [151, 160, 148]. In this work, we make effective use of a handful of interventional corrections provided by a single human operator to autonomously generate large-scale interventional data, substantially reducing the operator burden.

Imitation Learning from Human Demonstrations. Behavioral Cloning (BC) [204] on demonstrations collected using robot teleoperation with human operators has shown remarkable performance in solving real-world robot manipulation tasks [277, 161, 162, 27, 109, 2]. However, scaling this paradigm can be costly due to the need for large amounts of data, requiring many hours of human operator time [27, 109, 152]. Furthermore, policies trained via IL are often brittle and can fail when deployment conditions change from the training data [215].

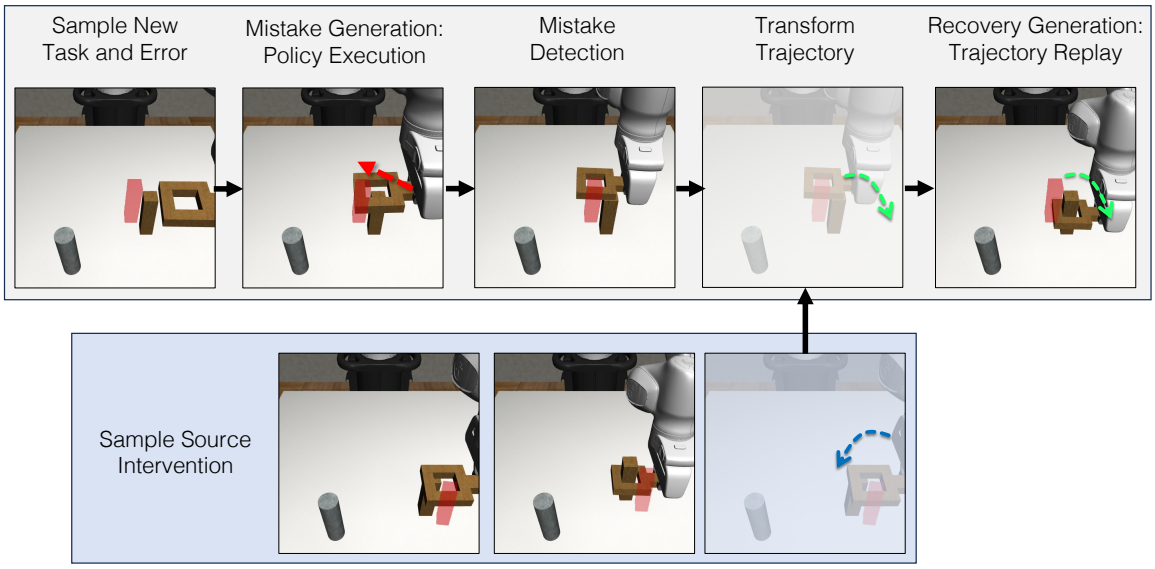


Figure 4.2: **I-Gen Data Generation Example.** We provide an example of how I-Gen generates a new intervention. First, a new task instance is sampled with a new configuration (square peg location) and observation corruption (incorrect peg location highlighted in red). We execute the robot policy to generate mistake behavior for the new task instance. When a mistake is detected, we sample a human intervention segment from the source dataset and transform it to adapt to the current scene. Finally, we executed the transformed recovery segment in the environment.

Interactive Imitation Learning. Interactive IL allows demonstrators to provide corrective supervision in situations where policies require assistance. Some approaches require an expert to relabel states encountered by the agent with actions that the expert would have taken [215, 42], but it can be difficult for human supervisors to relabel robot actions in hindsight [137]. An alternative is to cede control of the system to a human supervisor for short corrective trajectories (termed *interventions*) in states where the robot policy needs assistance. Interventional data collection can either be human-gated [125, 151], where the human monitors the policy and decides when to provide interventions, or robot-gated [98, 97, 99], where the robot decides when the human should provide interventions. However, these approaches require collecting a sufficient number of human interventions for the robot to learn robust recovery. In this work, we develop a novel data generation mechanism based on replay-based imitation [159, 263, 114] in order to alleviate this burden.

Policy Adaptation under Domain Shift. There are other approaches besides interactive IL for increasing policy robustness. These include injecting noise during demonstration collection [138], having human operators intentionally introduce mistakes and corrections during data collection [24], and enabling policies to deal with partial observability [181, 48]. Other approaches include employing a planner to return to states that the agent has seen

before [267, 52], using Reinforcement Learning (RL) with learned rewards to help an agent adapt to new object distributions [91], and using counterfactual data augmentation to identify irrelevant concepts and ensure agent behavior will not be affected by them [196]. There are also approaches to make policies trained with RL more robust, such as domain randomization [250, 197], using adversarial perturbations [164], and training agents to recover from unsafe situations [247].

MimicGen. MimicGen [159] is a recently proposed system for automatically generating task demonstrations via trajectory adaptation via leveraing known object poses. I-Gen employs a similar mechanism for synthesizing trajectories but has several key differences. Unlike MimicGen, I-Gen (1) generates interventional data rather than full demonstrations, (2) relaxes the assumption of precise object pose knowledge, which is critical to MimicGen's success, (3) integrates closed-loop policy execution that allows the robot to visit novel states during the data generation process, and (4) allows variation in not just object poses but also robot belief states about these object poses.

4.3 Preliminaries

Problem Statement. We model the task environment as a Partially Observable Markov Decision Process (POMDP) with state space S, observation space O, and action space A. The robot does not have access to the transition dynamics or reward function but has a dataset of samples $D = \{(o, a)\}_{i=1}^{N}$ from an expert human policy $\pi_H : O \to A$. We assume that while the human observes observation o, the robot's observation is corrupted by some function z, yielding $z(o) = o' \in O$ (e.g., due to sensor noise or network delay). In this work we train policies on demonstration datasets D using supervised learning with the objective $\arg\min_\theta \mathbb{E}_{(o,a)\sim D}[-\log \pi_\theta(a|o)]$.

Assumptions. I-Gen has assumptions similar to MimicGen [159]. **(Assumption 1)** the action space consists of delta-pose commands in Cartesian end effector space; **(Assumption 2)** the task is a known sequence of object-centric subtasks; **(Assumption 3)** object poses can be observed at the beginning of each subtask during data collection (but not deployment). **(Assumption 4)** We also assume that demonstrated recovery behavior can be explained by some component of the robot's observations $\{o'_1, o'_2, \dots\}$ during a human intervention despite corruption by z. Without this assumption, it would not be possible for the robot to learn a policy that maps o' to $\pi_H(o)$. This information can be provided, for instance, in additional observation modalities such as force-torque sensing or tactile sensing that provide a coarse signal about an object's pose. Some settings may not require any additional information: for example, a fully closed gripper can inform the robot it must recover from a missed grasp.

MimicGen Data Generation System. MimicGen [159] takes a small set of source human demonstrations D_{src} and uses it to automatically generate a large dataset D in a target environment. It first divides each source trajectory $\tau \in D_{src}$ into object-centric manipulation segments $\{\tau_i\}_{i=1}^{M}$, each of which corresponds to an object-centric subtask (Assumption 2 above). Each segment is a sequence of end effector poses. Then, to generate a demon-

stration in a new scene, it uses the pose of the object corresponding to the current subtask, and transforms the poses in a source human segment τ_i (with an SE(3) transform) such that the relative poses between the end effector and the object frame are preserved between the source demonstration and the new scene. It also adds an interpolation segment between the robot's current configuration and the start of the transformed segment. Then, the sequence of poses in the interpolation segment and transformed segment are executed by the robot end effector controller open-loop until the current subtask is complete, at which point the process repeats for the next subtask. We use a data generation mechanism similar to MimicGen to generate intervention trajectory segments in Section 4.4.

4.4 IntervenGen

Algorithm 2 displays the full pseudocode for IntervenGen. It takes as input the initial state distribution p_0, a base dataset of demonstrations D, and three hyperparameters k, m, n. On each of one or more iterations, the system: (1) trains a policy π_θ on the current dataset; (2) rolls out π_θ for interventional data collection with human teleoperation; (3) synthesizes new interventions with closed-loop policy execution and open-loop trajectory replay; (4) returns the new synthetic dataset.

Interventional Data Collection

We consider human-gated interventions [125], in which the human monitors the robot policy execution and intermittently takes control to correct policy mistakes. As in DAgger [215], this enables the human to demonstrate corrective recovery behavior from mistakes made by the robot policy that otherwise would not be visited in full human task demonstrations (due to distribution shift). The base robot policy π_θ executed during interventional data collection can come from anywhere, but is typically initialized from behavior cloning on an initial set of offline task demonstrations D [97, 160, 215]. Each collected trajectory can be coarsely divided into robot-generated "mistake" segments and human-generated "recovery" segments.

Mistake Generation: Closed-Loop Policy Execution

We aim to use the collected human interventions to automatically synthesize interventions for new scene configurations. Recall that, in prior work, MimicGen generates data by executing a sequence of object-centric trajectories in an open-loop manner. In contrast, an appealing property of our interventional IL setting is access to the robot policy π_θ that is executed during interventional data collection with the human operator.

We use this robot policy during the *data generation* process to broaden the distribution of visited mistake states. Unlike MimicGen, we can execute the policy in the new scene configuration. This has two benefits: (1) rather than assuming the policy will fail in the

same manner as the source trajectory, the generated mistake will reflect the genuine behavior of the policy in the new configuration, and (2) it becomes possible to generate new mistake trajectories for new corruptions of the observed object poses. For example, if sensor noise corrupts the object pose during interventional data collection, a new noise corruption can be applied during the data generation process. This allows data diversity in both object poses and the robot's erroneous beliefs about where the objects are (see Fig. 4.2). The use of policy execution during data generation requires that we know when to terminate the policy execution. In our experiments, we use contact detection to determine whether or not the policy made a mistake. A more flexible option could be to use a learned classifier or robot-gated intervention criteria such as ThriftyDAgger [97].

Recovery Generation: Open-Loop Trajectory Replay

In each episode of synthetic data generation, once we have completed policy execution and entered a new mistake state, we generate a recovery trajectory. We select a random source trajectory, segment out the human recovery portion of the trajectory, and adapt the trajectory to the current environment state. This adaptation consists of (1) transforming the source trajectory to the current object pose, (2) linearly interpolating in end-effector space to the beginning of the transformed trajectory, and (3) executing the transformed trajectory open-loop (see Fig. 4.2). Note that each object-centric subtask in a single task instance can have zero, one, or multiple instances of alternating between mistake and recovery.

Output Filtering and Dataset Aggregation

It is possible that the executed trajectory may not complete the task successfully. For instance, the recovery trajectory may be unable to recover from the new mistake state reached by the robot. Consequently, we only keep the generated demonstration if it successfully completes the task. We also filter out the segment of the synthetic demonstration that corresponds to the human recovery segment; such filtering is used by common algorithms such as DAgger [215] and HG-DAgger [125] and can prevent the imitation of mistakes. Each filtered episode of synthetic data is aggregated into the base dataset D (used to train the base policy π_θ), and the policy is retrained on the new dataset after data generation. If desired, the entire process of data collection, data generation, and policy training can be iterated.

Inter-Subtask Recovery and Offline Mode

The I-Gen framework accommodates additional modules not considered in the main set of experiments that greatly increases its range of applications, including (1) policy recovery from more severe failure modes that revert to earlier subtasks and (2) "offline" I-Gen, which allows humans to demonstrate mistakes intentionally [24]. We include experiments for these modules on the supplemental website.

Algorithm 2 IntervenGen

Require: Initial state distribution p_0, base dataset D
Require: Number of iterations k, human intervention episodes m, and synthesized trajectories n

procedure I-GEN($p_0, D; k, m, n$)
  for $i \in [1, ..., k]$ **do** ▷ One or more iterations
    $\pi_\theta \leftarrow$ TRAIN-POLICY(D)
    $\mathcal{D} = \emptyset$
    for $j \in [1, ..., m]$ **do** ▷ **Data Collection**
      $s_0 \sim p_0$ ▷ Sample initial state
      $\tau \leftarrow$ EXECUTE-POLICY(s_0, π_θ)
      INTERVENE(τ) ▷ Human intervention
      $\mathcal{D} \leftarrow \mathcal{D} \cup \tau$
    for $j \in [1, ..., n]$ **do** ▷ **Data Generation**
      $s_0 \sim p_0$
      $\xi \leftarrow$ EXECUTE-POLICY(s_0, π_θ)
      $t \leftarrow$ TERMINATE-POLICY(ξ)
      $\tau \sim \mathcal{D}$ ▷ Sample source demonstration
      $\tau \leftarrow \tau$[human] ▷ Filter intervention
      $\tau' \leftarrow$ ADAPT(ξ, τ) ▷ Transform trajectory
      $\xi \leftarrow \xi \oplus$ REPLAY(τ')
      if SATISFIES-GOAL($\xi[-1]$) **then**
        $D \leftarrow D \cup \xi[t:]$ ▷ Filter intervention
  return D

4.5 Experiment Setup

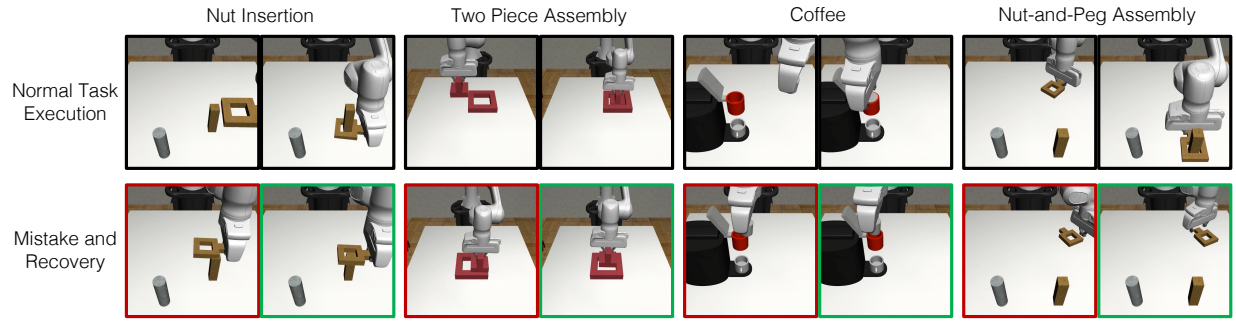


Figure 4.3: **Tasks.** We evaluate I-Gen in several contact-rich, high-precision tasks. The top row shows normal task execution while the bottom row shows typical mistakes encountered by the agent when using inaccurate object poses (or object geometry for Nut-and-Peg Assembly) and associated recovery behaviors.

We consider 4 tasks in the MuJoCo [251] robosuite simulation environment [280] (Fig. 4.3)

and 1 physical experiment. Each task involves contact-rich manipulation via continuous control. The tasks vary in object geometry, object pose, observation error, and number of manipulation stages.

Nut Insertion: The robot must place a square nut (held in-hand) onto a square peg. The peg position is sampled in a 10 cm x 10 cm region at the start of each episode.

2-Piece Assembly: The robot must place an object into a square receptacle with a narrow affordance region. The receptacle position is sampled in a 10 cm x 10 cm region at the start of each episode.

Coffee: The robot must place and release a coffee pod into a coffee machine pod holder with a narrow affordance region. The coffee machine position is sampled in a 10 cm x 10 cm region at the start of each episode.

Nut-and-Peg Assembly [280, 162]: A multi-stage task consisting of (1) grasping a nut with a varying initial position and orientation and (2) placing it on a peg in a fixed target location. The nut is placed in a 0.5 cm x 11.5 cm region with a random top-down rotation at the start of each episode.

Physical Block Grasp: A Franka robot arm must reach a block and grasp it. The initial block position is sampled in a 20 cm x 30 cm region at the start of each episode.

Sources of Observation Error. In most environments, the source of observation error is *sensor noise*: at test time, uniform random noise is applied to the observed position of the peg (± 4 cm in each dimension, with at least 2 cm in one dimension), receptacle (± 4 cm in each dimension, with at least 1 cm in one dimension), coffee machine (radial noise between 2 cm and 4 cm), and block (± 1 cm in x and ± 7 cm in y, with at least 2.5 cm in y) respectively. In the Nut-and-Peg Assembly environment, the source of observation error is *object geometry*: for an identical observed nut pose, the nut handle may exist on either of two sides of the nut. This setting corresponds to object model misspecification during pose registration.

Experimental Setup

Data Collection. For interventional data collection, we use the remote teleoperation system proposed by Mandlekar et al. [160]. The observation space consists of robot proprioception (6DOF end effector pose and gripper finger width) and object poses, while the action space consists of 6DOF pose deltas and a binary gripper open/close command (except for Block Grasp, which uses 3DOF position control with fixed rotation). For the base policy π_θ used in each task, we (1) collect 10 full human task demonstrations in each environment *without* observation corruption (i.e., ground truth poses), (2) synthesize 1000 demonstrations with MimicGen [159], and (3) train an off-the-shelf BC-RNN policy with default hyperparameters using the robomimic framework [162], with the exception of an increased learning rate of 0.001 [159].

Data Generation. We then deploy π_θ in the test environment *with* observation corruption (i.e., object pose error) and collect 10 human-gated interventions. These interventions are expanded to 1000 synthetic interventions with I-Gen and aggregated with the 1000

demonstrations used to train the base policy. Finally, we train a new BC-RNN policy on the aggregated dataset. We report policy performance as the success rate over 50 trials for the highest performing checkpoint during training (where training takes 2000 epochs with evaluation every 50 epochs), as in [162, 159].

Observability. In order for demonstrated recovery behavior to be learnable (Section 4.3), I-Gen and all baselines can access additional observation information in Nut Insertion, Two-Piece Assembly, Coffee, and Block Grasp upon contact between (1) the nut and peg, (2) object and receptacle, (3) pod and pod holder, and (4) gripper and cube, respectively. We study both the idealized case of full observability (i.e., ground truth pose) upon contact in Section 4.6 and partially improved observability (e.g., position of contact) in Section 4.6. These are intended to be surrogates for sensor modalities such as force-torque sensing that can help inform the robot about the object pose when its belief is wrong. For Nut-and-Peg Assembly, we do not add additional information, as a closed gripper state is sufficient for the policy to map a missed grasp to learned recovery.

Physical Experiment Setup. We wish to evaluate whether or not policies trained on simulation data from I-Gen can retain their robustness to erroneous state estimation when they are deployed directly in the real world. To do this, we train a policy for the Block Grasp task in simulation and deploy it zero-shot on a physical robot. We use a Franka Research 3 robot arm and gripper and a red cube with a side length of 5 cm. We use an Intel RealSense D415 depth camera and Iterative Closest Point (ICP) for cube pose estimation. The deployed policies output continuous control delta-pose actions at 20 Hz and do not require any real-world data or fine-tuning. See Figure 4.4 for images of the transfer process.

Baselines

We implement and evaluate the following baselines. Each baseline corresponds to a *different dataset* used to train the agent (all agents are trained with BC-RNN [162]):

Base: Deploy the base policy in the test environment without any additional data or fine-tuning.

Source Interventions (Source Int): Deploy the base policy π_θ, collect 10 human interventions when the policy makes mistakes, and add them to the base dataset.

Weighted Source Interventions (Weighted Src Int) [160]: Same as Source Interventions, but weight the intervention data higher so that it is sampled as frequently as the base data despite its smaller quantity.

Source Demonstrations (Source Demo): Collect 10 full human task demonstrations in the test environment.

MimicGen Demonstrations (MG Demo) [159]: Same as Source Demonstrations, but use (regular) MimicGen to generate 1000 synthetic demonstrations from the initial 10.

Policy Execution Ablation (I-Gen - Policy): Augment the 10 source interventions to 1000 I-Gen interventions, but do not use policy execution to generate new mistake states.

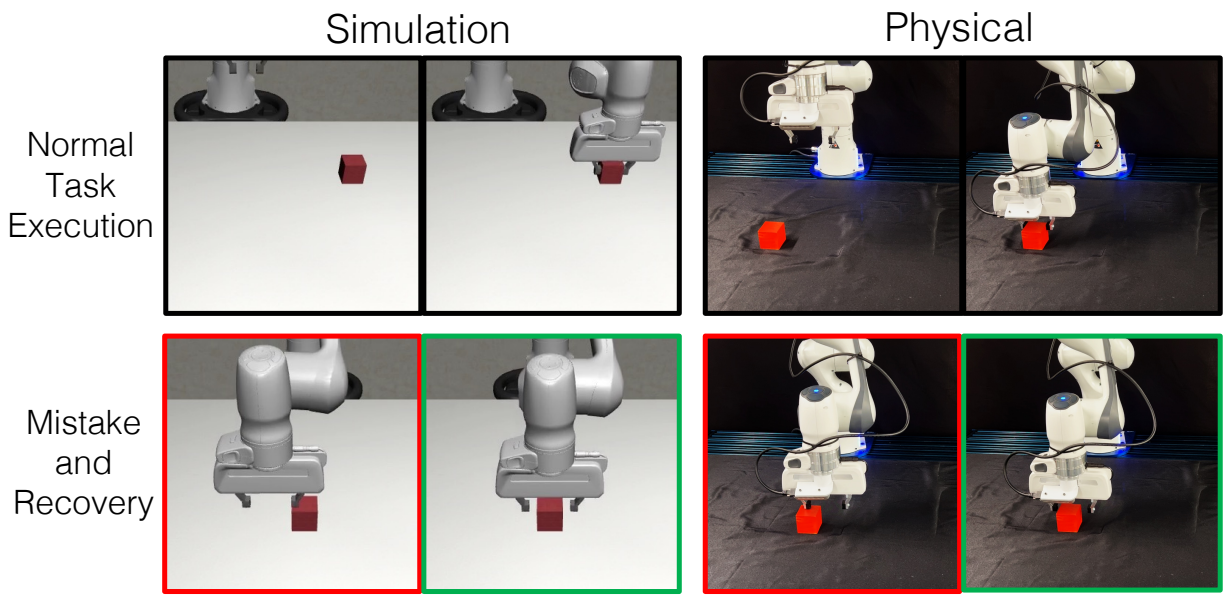


Figure 4.4: **Sim-to-Real.** We evaluate sim-to-real transfer for a block grasping task with a Franka Panda robot. Similar to Figure 4.3 we show normal task execution, typical mistakes due to inaccurate object poses, and associated recovery for the simulation and real world environments. The results show that I-Gen can facilitate sim-to-real transfer of learned control policies, and that these policies retain robustness to erroneous perception.

4.6 Experiments

In this section, we summarize the key takeaways from the comparisons presented in Tables 4.1 and 4.2.

I-Gen vastly improves policy robustness under pose estimation error. In Table 4.1, we observe that I-Gen improves policy performance by 3.5×, 10.7×, and 39× over the base policy in Nut Insertion, 2-Piece Assembly, and Coffee respectively, despite only collecting 10 human interventions.

I-Gen significantly improves upon naïve uses of an equivalent amount of full human demonstration data. I-Gen consistently outperforms human demonstrations collected at test time (Source Demo, Table 4.1) by 56%-68%. Even if these demonstrations are expanded by 100× with MimicGen (MG Demo), I-Gen still outperforms by 34%-62%. Since the human's observability does not match the robot's, the human can teleoperate toward the true object poses. Thus, the robot does not observe any recovery behavior in the offline data.

I-Gen significantly improves upon naïve uses of an equivalent amount of interventional human data. Source Int in Table 4.1 underperforms I-Gen by 58%-70%.

While helpful, with only 10 human interventions, the data is insufficient to learn robust recovery under pose error. This remains the case even if the intervention data is weighted higher, in which case the agent overfits to the 10 interventions and underperforms I-Gen by 48%-74%. With the same budget of interventional human data, I-Gen can generate much richer coverage of the distribution of mistakes under the base policy.

I-Gen significantly improves upon naïve uses of MimicGen. We observe a significant 34%-62% improvement over MimicGen on full task demonstrations (MG Demo, Table 4.1). We also observe that the policy execution component (Section 4.4) boosts performance by 12%-38% respectively over the ablation, indicating that expanding the mistake distribution is valuable. While the ablation dataset covers variation in the object pose, it does not cover variation in the error; only the 10 mistake segments in the source dataset are available. This shows that the novel components we introduced in I-Gen are crucial for high performance.

I-Gen is useful across different environments. While 2-Piece Assembly and Coffee have narrower tolerance regions than Nut Insertion that lower success rates across the board (16%-20% for the base policy, 30%-48% for other baselines, and 18%-28% for I-Gen), the relative performance of I-Gen remains consistent across environments: I-Gen outperforms all baselines by 12%-76% in Nut Insertion, 18%-64% in 2-Pc Assembly, and 38%-78% in Coffee.

I-Gen is useful across different sources of observation error. Results for the Nut-and-Peg Assembly task with object geometry error are in Table 4.2. We evaluate each policy with 50 evaluations of each of the two possible geometries. Base and Source Int attain perfect performance on the original geometry but struggle with the alternate geometry (0%-6% performance). MG Demo has the opposite issue: since it consists of test-time demonstrations with the alternate geometry, it can attain perfect performance on the alternate but 0% on the original. A mixture of full demonstrations on both geometries (Base + MG Demo) attains an even 60% and 64%; since it does not observe recovery behavior it must guess between the two object geometries and has difficulty performing much higher than the 50% expected value of random chance. Finally, I-Gen maintains 92% performance on the original geometry but also learns to recover when missing its grasp due to the alternate geometry (88%), leading to a 28%-40% improvement in the average case over baselines. See the website for videos.

I-Gen facilitates sim-to-real transfer of learned control policies, and these policies retain robustness to erroneous state estimation. In Table 4.4 we observe that state-based policies for the Block Grasp task deployed zero-shot on the physical system perform similarly to simulation. By improving robustness to incorrect pose estimation, I-Gen facilitates sim-to-real transfer for state-based policies, which are easier to transfer across visual domain gaps than image-based policies but rely on accurate perception. I-Gen outperforms baselines by 14%-94% in simulation and 30%-90% in real world trials, suggesting learned recovery behaviors can transfer to real. The policy is also robust to physical perturbations, dynamic object pose changes, and visual distractors; see the website for videos.

Dataset	Nut Insertion	2-Pc Assembly	Coffee
Base	22%	6%	2%
Source Int	40%	6%	10%
Weighted Src Int [160]	50%	16%	6%
Source Demo	42%	12%	12%
MG Demo [159]	64%	16%	18%
I-Gen - Policy (Ours)	86%	52%	42%
I-Gen (Ours)	**98%**	**70%**	**80%**

Table 4.1: Results in three simulation domains with noisy pose estimation and full observability upon contact. I-Gen outperforms baselines across environments.

Dataset	Geometry 1	Geometry 2	Mixture
Base	**100%**	0%	50%
Source Int	**100%**	6%	53%
MG Demo [159]	0%	**100%**	50%
Base + MG Demo	64%	60%	62%
I-Gen	92%	88%	**90%**

Table 4.2: Results in the Nut-and-Peg Assembly experiment. While baselines typically overfit to one geometry or struggle with disambiguating the two, I-Gen attains high performance on the mixture of geometries.

Analysis

In this section, we present further analysis on various aspects of I-Gen.

How is agent performance affected as observability decreases? For Nut Insertion, we replace true pose information upon contact with the mean position of the first contact between the nut and peg; for 2-Piece Assembly, we provide the unit vector in the direction of the true pose at the first point of contact. Table 4.3 in comparison with Table 4.1 shows that, as expected, a degradation in observability results in a degradation in agent performance. However, I-Gen performance falls by only 4%-8%, indicating partial observability can be sufficient to ground recovery behavior. An important direction for future work is investigating raw real-world perception signals such as force-torque sensing.

How does performance vary across training seeds? I-Gen in the (full observability) Nut Assembly task attains 98%, 100%, and 98% for 3 training seeds, indicating stability across runs (more evidence on supplemental website).

How does synthetic IntervenGen data compare to an equal amount of human data? In 2-Piece Assembly, 100 I-Gen interventions (from 10 human interventions) attain 24% while 100 human interventions attain 46%. Both improve upon 10 human interventions, which only attains 6% (Table 4.1). However, 1000 I-Gen interventions from 10 human

Dataset	Nut Insertion	2-Pc Assembly
Base	26%	6%
Source Int	40%	6%
MG Demo [159]	46%	22%
I-Gen - Policy	68%	42%
I-Gen	**90%**	**66%**

Table 4.3: Additional evaluation in two domains with partially improved (rather than full) observability upon contact.

Dataset	Simulation	Real
Base	6%	0%
Source Int	26%	10%
MG Demo [159]	42%	50%
I-Gen - Policy	86%	60%
I-Gen	**100%**	**90%**

Table 4.4: Sim-to-real results for the block grasping task in simulation (50 trials) and zero-shot evaluation of these policies in the real world (10 trials).

interventions (70%) can outperform 100 human interventions, and 100 human interventions take significantly more human time and effort to collect than 10 human interventions (29.9 minutes instead of 3.6 minutes).

How does performance scale with the amount of synthetically generated interventions? With the same 10 human source interventions in 2-Piece Assembly, an agent trained on 200 synthetic I-Gen interventions attains 34%, 1000 interventions attains 70% (Table 4.1), and 5000 interventions attains 88%. This suggests performance scales with dataset size, at the cost of additional data generation time.

4.7 Conclusion

We present IntervenGen (I-Gen), a data generation system for corrective interventions that cover a large distribution of policy mistakes given a small number of source human interventions. We show that training on synthetic data generated by I-Gen compares favorably to collecting more human demonstrations and interventions in terms of both policy performance and human effort.

Although I-Gen improves on MimicGen and reduces its reliance on accurate pose estimation, I-Gen shares some of its limitations. Specifically, we consider only quasi-static tasks with rigid body objects, and we assume valid interventions can be synthesized by transforming source trajectory data.

Future work involves applying I-Gen in settings with force-torque sensing to improve behavioral adaptation for contact-rich and high-precision tasks. I-Gen can also be used to rapidly adapt policy behavior toward individual human preferences over how a manipulation task should be carried out without extensive data collection. Finally, I-Gen can also be applied to facilitating sim-to-real transfer of IL policies by acting as a domain randomization [250] procedure. Namely, while RL algorithms can autonomously learn adaptations to dynamical domain randomization, IL typically requires generating new human behavior for these variations. I-Gen may dramatically reduce the data requirements and enable policies to deal with such variations with only a handful of corrective behaviors.

Part II

Interactive Fleet Learning

Chapter 5

Fleet-DAgger: Interactive Robot Fleet Learning

Part II extends the interactive IL of Part I to the fleet setting of multiple robots and multiple humans. This chapter formalizes this setting as interactive fleet learning (IFL) and introduces new IFL algorithms, benchmarks, and experiments.

5.1 Introduction

Amazon, Nimble, Plus One, Waymo, and Zoox use remote human supervision of robot fleets in applications ranging from self-driving taxis to automated warehouse fulfillment [28, 240, 149, 154, 50]. These robots intermittently cede control during task execution to remote human supervisors for corrective interventions. The interventions take place either during learning, when they are used to improve the robot policy, or during execution, when the policy is no longer updated but robots can still request human assistance when needed to improve reliability. In the *continual learning* setting, these occur simultaneously: the robot policy has been deployed but continues to be updated indefinitely with additional intervention data. Furthermore, any individual robot can share its intervention data with the rest of the fleet. As opposed to robot swarms that must coordinate with each other to achieve a common objective, a robot *fleet* is a set of independent robots simultaneously executing the same control policy in parallel environments. We refer to the setting of a robot fleet learning via interactive requests for human supervision (see Figure 5.1) as *Interactive Fleet Learning (IFL)*.

Of central importance in IFL is the supervisor allocation problem: how should limited human supervision be allocated to robots in a manner that maximizes the throughput of the fleet? Prior work studies this in the single-robot, single-human case. A variety of interactive learning algorithms have been proposed that estimate quantities such as uncertainty [171], novelty [127, 139, 97], risk [139, 97], and predicted action discrepancy [276, 98]. However, it remains unclear which algorithms are the most effective when generalized to the multi-robot,

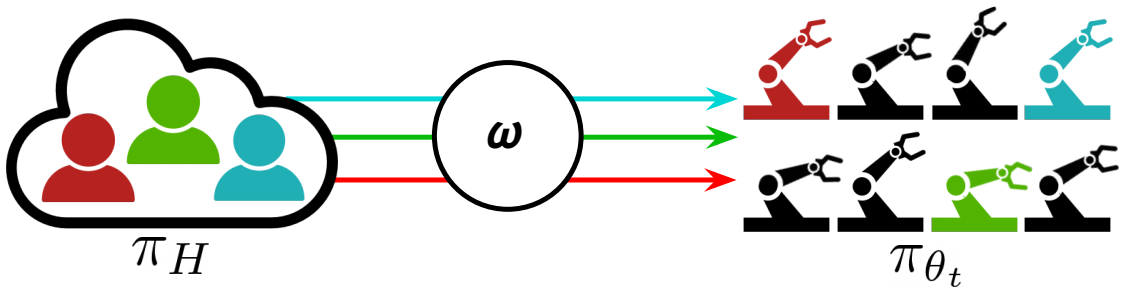


Figure 5.1: In the *Interactive Fleet Learning (IFL)* setting, a set of M remote human supervisors are allocated to a fleet of N robots ($N \gg M$) with a robot-gated allocation policy ω. The humans share control policy π_H and the robot fleet shares control policy π_{θ_t}, which learns from new human intervention data over time (Section 3).

multi-human case.

To this end, we formalize the IFL problem and present the IFL Benchmark (IFLB), a new open-source Python toolkit and benchmark for developing and evaluating human-to-robot allocation algorithms for fleet learning. The IFLB includes environments from Isaac Gym [157], which enabled efficient simulation of thousands of learning robots for the first time in 2021. This paper makes the following contributions: (1) the first formalism for multi-robot, multi-human interactive learning, (2) the Return on Human Effort (ROHE) metric for evaluating IFL algorithms, (3) the IFLB, an open-source software benchmark and toolkit for IFL algorithms with 3 Isaac Gym environments for complex robotic tasks, (4) Fleet-DAgger, a novel family of IFL algorithms for supervisor allocation, (5) results from large-scale simulation experiments with a fleet of 100 robots, and (6) real robot results with 4 physical robot arms and 2 human supervisors providing teleoperation remotely over the Internet.

5.2 Related Work

Allocating Human Supervisors to Robots at Execution Time

For human-robot teams, deciding when to transfer control between robots and humans during execution is a widely studied topic in the literature of both sliding autonomy [229, 228, 67] and Human-Robot Interaction (HRI). In sliding autonomy, also known as adjustable autonomy [224, 132] or adaptive automation [234], humans and robots dynamically adjust their level of autonomy and transfer control to each other during execution [67, 234]. Since identifying which robot to assist in a large robot fleet can be overwhelming for a human operator [37, 142, 45, 200, 46], several strategies, such as using a cost-benefit analysis to decide whether to request operator assistance [229] and using an advising agent to filter robot requests [213], have been proposed to improve the performance of human-robot teams [200,

213, 57] and increase the number of robots that can be controlled [274], a quantity known as "fan-out" [184]. Other examples include user modeling [229, 228, 200, 57] and studying interaction modes [32] for better system and interface design [6, 36]. Zheng et al. [279] propose computing the estimated time until stopping for mobile robots and prioritizing robots accordingly. Ji et al. [112] consider the setting where physical assistance is required to resume tasks for navigation robots and formalize single-human, multi-robot allocation as graph traversal. Dahiya et al. [62] formulate the problem of multi-human, multi-robot allocation during execution as a Restless Multi-Armed Bandit problem. Allocation of humans to robots has also been studied from the perspectives of queueing theory and scheduling theory [46, 56, 230, 84, 216, 65]. The vast majority of the human-robot teaming and queueing theory work, however, does not involve learning; the robot control policies are assumed to be fixed. In contrast, we study supervisor allocation during robot learning, where allocation affects not only human burden and task performance but also the efficiency of policy learning.

Single-Robot, Single-Human Interactive Learning

Imitation learning (IL) is a paradigm of robot learning in which a robot uses demonstrations from a human to initialize and/or improve its policy [15, 253, 54, 203, 95, 12]. However, learning from purely offline data often suffers from distribution shift [215, 138], as compounding approximation error leads to states that were not visited by the human. This can be mitigated with online data collection with algorithms such as Dataset Aggregration (DAgger) [215] and interactive imitation learning [42, 110, 211]. Human-gated interactive IL algorithms [125, 241, 160] require the human to monitor the robot learning process and decide when to take and cede control of the system. While intuitive, these approaches are not scalable to large fleets of robots or the long periods of time involved in continual learning, as humans cannot effectively focus on many robots simultaneously [45, 200, 46] and are prone to fatigue [177]. To reduce the burden on the supervisor, several robot-gated interactive IL algorithms such as SafeDAgger [276], EnsembleDAgger [171], LazyDAgger [98], and ThriftyDAgger [97] have been proposed, in which the robot actively solicits human interventions when certain criteria are met. Interactive reinforcement learning (RL) [271, 135, 261, 116, 254] is another active area of research in which robots learn from both online human feedback and their own experience. However, these interactive learning algorithms are designed for and primarily studied in the single-robot, single-human setting. Other works related to single-robot interactive learning include task allocation [259]; in contrast, we focus on efficient robot learning of a single control policy.

Multi-Robot Interactive Learning

In this paper, we study allocation policies for multiple humans and multiple robots. While many existing works [165, 270, 72, 198, 219] have leveraged NVIDIA's Isaac Gym's [157] capability of parallel simulation to accelerate reinforcement learning with multiple robots, these approaches do not involve human supervision. The work that is closest to ours is

by Swamy et al. [245], who study the multi-robot, single-human problem of allocating the attention of one human operator during robot fleet learning. They propose to learn an internal model of human preferences as a human supervises a small fleet of 4 robots and use this model to assist the human in supervising a larger fleet of 12 robots. While this approach mitigates the scaling issue in human-gated interactive IL, even a small fleet of robots can be difficult for a single human supervisor to simultaneously monitor and control.

To the best of our knowledge, this work is the first to formalize and study multi-robot, multi-human interactive learning. This problem setting poses unique challenges, especially as the size of the fleet grows large relative to the number of humans, as each human allocation affects both the robot that receives supervision and the robots that do not receive human attention.

5.3 Interactive Fleet Learning Problem Formulation

We consider a fleet of N robots operating in parallel as a set of N independent Markov decision processes (MDPs) $\{\mathcal{M}_i\}_{i=1}^N$ specified by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, p_i^0)$ with the same state space $\mathcal{S}$, action space $\mathcal{A}$, unknown transition dynamics $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1)$, but potentially different initial state distributions p_i^0. We assume the MDPs have an identical indicator function $c(s) : \mathcal{S} \to \{0, 1\}$ that identifies which states $s \in \mathcal{S}$ violate a *constraint* in the MDP. States that violate MDP constraints are fault states from which the robot cannot make further progress. For instance, the robot may be stuck on the side of the road or have incurred hardware damage. We assume that the timesteps are synchronized across all robots and that they share the same non-stationary policy $\pi_{\theta_t} : \mathcal{S} \to \mathcal{A}$, parameterized by θ_t at each timestep t.

The collection of $\{\mathcal{M}_i\}_{i=1}^N$ can be reformulated as a single MDP $\mathcal{M} = (\mathcal{S}^N, \mathcal{A}^N, \bar{p}, \bar{r}, \gamma, \bar{p}^0)$, composed of vectorized states and actions of all robots in the fleet (denoted by bold font) and joint transition dynamics. In particular, $\mathbf{s} = (s_1, ..., s_N) \in \mathcal{S}^N$, $\mathbf{a} = (a_1, ..., a_N) \in \mathcal{A}^N$, $\bar{p}(\mathbf{s}^{t+1}|\mathbf{s}^t, \mathbf{a}^t) = \Pi_{i=1}^N p(s_i^{t+1}|s_i^t, a_i^t)$, $\bar{r}(\mathbf{s}, \mathbf{a}) = \Sigma_{i=1}^N r(s_i, a_i)$, and $\bar{p}^0 = \Pi_{i=1}^N p_i^0(s_i^0)$.

We assume that robots can query a set of $M \ll N$ human supervisors for assistance interactively (i.e., during execution of π_{θ_t}). We assume that each human can help only one robot at a time and that all humans have the same policy $\pi_H : \mathcal{S} \to \mathcal{A}_H$, where $\mathcal{A}_H = \mathcal{A} \cup \{R\}$ and R is a *hard reset*, an action that resets the MDP to the initial state distribution $s^0 \sim p_i^0$. As opposed to a *soft reset* that can be performed autonomously by the robot via a reset action $r \in \mathcal{A}$ (e.g., a new bin arrives in an assembly line), a *hard reset* requires human intervention due to constraint violation (i.e., entering some s where $c(s) = 1$). A human assigned to a robot either performs hard reset R (if $c(s) = 1$) or teleoperates the robot system with policy π_H (if $c(s) = 0$). A hard reset R takes t_R timesteps to perform, and all other actions take 1 timestep.

Supervisor allocation (i.e., the assignment of humans to robots) is determined by an

allocation policy

$$\omega : (\mathbf{s}^t, \pi_{\theta_t}, \boldsymbol{\alpha}^{t-1}, \mathbf{x}^t) \mapsto \boldsymbol{\alpha}^t \in \{0,1\}^{N \times M} \quad \text{s.t.} \quad \sum_{j=1}^{M} \boldsymbol{\alpha}_{ij}^t \leq 1 \text{ and } \sum_{i=1}^{N} \boldsymbol{\alpha}_{ij}^t \leq 1 \quad \forall i, j, \quad (5.1)$$

where $\mathbf{s}^t$ are the current states for each of the robots, $\boldsymbol{\alpha}^t$ is an $N \times M$ binary matrix that indicates which robots will receive assistance from which human at the current timestep t, and $\mathbf{x}^t$ is an augmented state containing any auxiliary information for each robot, such as the type and duration of an ongoing intervention. Unlike Dahiya et al. [62] which studies execution-time allocation, the allocation policy ω here depends on the current robot policy π_{θ_t}, which in turn affects the speed of the policy learning. While there are a variety of potential objectives to consider, e.g., minimizing constraint violations in a safety-critical environment, we define the IFL objective as *return on human effort (ROHE)*:

$$\max_{\omega \in \Omega} \mathbb{E}_{\tau \sim p_{\omega, \theta_0}(\tau)} \left[\frac{M}{N} \cdot \frac{\sum_{t=0}^{T} \bar{r}(\mathbf{s}^t, \mathbf{a}^t)}{1 + \sum_{t=0}^{T} \|\omega(\mathbf{s}^t, \pi_{\theta_t}, \boldsymbol{\alpha}^{t-1}, \mathbf{x}^t)\|_F^2} \right], \quad (5.2)$$

where Ω is the set of allocation policies, T is the total amount of time the fleet operates (rather than the time horizon of an individual task execution), θ_0 are the initial parameters of the robot policy, and $\|\cdot\|_F$ is the Frobenius norm. The objective is the expected ratio of the cumulative reward across all timesteps and all robots to the total amount of human time spent helping robots with allocation policy ω, with a scaling factor to normalize for the number of robots and humans and an addition of 1 in the denominator for the degenerate case of zero human time. Intuitively, the ROHE measures the performance of the robot fleet normalized by the total human effort required to achieve this performance. We provide a more thorough derivation of the ROHE objective in Appendix C.1.

Since human teleoperation with π_H provides additional online data, this data can be used to update the robot policy π_{θ_t} with some policy update function f (e.g., gradient descent):

$$\begin{cases} D^{t+1} \leftarrow D^t \cup D_H^t \text{ where } D_H^t \coloneqq \{(s_i^t, \pi_H(s_i^t)) : \pi_H(s_i^t) \neq R \text{ and } \sum_{j=1}^{M} \boldsymbol{\alpha}_{ij}^t = 1\} \\ \pi_{\theta_{t+1}} \leftarrow f(\pi_{\theta_t}, D^{t+1}) \end{cases} \quad (5.3)$$

5.4 Interactive Fleet Learning Algorithms

Fleet-DAgger

Given the problem formulation above, we propose Fleet-DAgger, a family of IFL algorithms, where an *IFL algorithm* is a supervisor allocation strategy (i.e., it specifies an $\omega \in \Omega$ as defined in Section 5.3). The learning algorithm in Fleet-DAgger is interactive imitation learning with dataset aggregation from prior work [215, 125, 97]: its policy update function f is supervised learning on D^t, which consists of all human data collected so far (Section 5.3). The novel component of each Fleet-DAgger algorithm is its supervisor allocation scheme based on

unique *priority function* $\hat{p} : (s, \pi_{\theta_t}) \to [0, \infty)$ that indicates a priority score to assign to each robot based on its state s and the current policy π_{θ_t}, where, similar to scheduling theory, a higher value indicates a higher priority robot. To reduce thrashing [98, 97], Fleet-DAgger algorithms also specify t_T, the minimum time a human supervisor must spend teleoperating a robot.

Fleet-DAgger uses priority function $\hat{p}$ and t_T to define an allocation ω as follows (see Appendix C.2 for the full pseudocode). At each timestep t, Fleet-DAgger first scores all robots with $\hat{p}$ and sorts the robots by their priority values. If a human supervisor is currently performing hard reset action R and t_R timesteps have not elapsed, that human continues to help that robot. If a human is currently teleoperating a robot and the minimum t_T timesteps have not elapsed, that human continues to teleoperate the robot. If a robot with a human supervisor continues to be high priority after the minimum intervention time (t_R for a hard reset or t_T for teleoperation) has elapsed, that human remains assigned to the robot. If a human is available to help a robot, the human is reassigned to the robot with the highest priority value that is currently unassisted. Finally, if a robot has priority $\hat{p}(\cdot) = 0$, it does not receive assistance even if a human is available.

Fleet-DAgger Algorithms

All algorithms below specify a unique priority function $\hat{p}$, which is synthesized with Fleet-DAgger as described in Section 5.4 to specify an allocation ω. More details are available in the appendix.

Constraint (C): The Constraint baseline measures the performance of the robot fleet when only trained on offline human demonstrations (i.e., $\forall t,\ \pi_{\theta_t} = \pi_{\theta_0}$). At all timesteps t, this baseline gives priority $\hat{p}(\cdot) = 1$ for robots that have violated a constraint ($c(s_i^t) = 1$) and require a hard reset, and $\hat{p}(\cdot) = 0$ for all other robots. We refer to this as C-prioritization for Constraint. Thus, the robot fleet can only receive hard resets from human supervisors (no human teleoperation). Without C-prioritization, robots that require hard resets would remain indefinitely idle.

Random: This baseline simply assigns a random priority for each robot at each timestep. To control the total amount of human supervision, we introduce a threshold hyperparameter such that if a robot's priority value is below the threshold, its priority is set to zero and it will not request help.

Fleet-EnsembleDAgger (U.C.): This baseline adapts EnsembleDAgger [171] to the IFL setting. EnsembleDAgger uses the output variance among an ensemble of neural networks bootstrapped on subsets of the training data as an estimate of epistemic uncertainty; accordingly, we define the robot priority for Fleet-EnsembleDAgger as ensemble variance. Since ensemble variance is designed for continuous action spaces, for environments with discrete action spaces we instead estimate the uncertainty with the Shannon entropy [232] among the outputs of a single classifier network.We refer to this priority function as U-prioritization for Uncertainty. Finally, since EnsembleDAgger was not designed for environments with constraint violations and idle robots will negatively affect the ROHE, we

add C-prioritization for a more fair comparison. Specifically, given an uncertainty threshold value, robots with uncertainty above threshold are prioritized first in order of their uncertainty (U), followed by constraint-violating robots (C).

Fleet-ThriftyDAgger (U.G.C.): This baseline adapts the ThriftyDAgger algorithm [97] to the IFL setting. ThriftyDAgger uses a synthesis of uncertainty (which we refer to as the U-prioritization value) and the probability of task failure (G-prioritization, estimated with a Goal critic Q-function) to query a human for supervision. Since Fleet-DAgger requires a single metric by which to compare different robots, we adapt ThriftyDAgger to the fleet setting by calculating a linear combination of the U value and G value after normalizing each value with running estimates of their means and standard deviations. As in [97], we pretrain the goal critic on an offline dataset of human and robot task execution. Similar to Fleet-EnsembleDAgger, we first prioritize by the combined uncertainty-goal value above a parameterized threshold, followed by C-prioritization.

Constraint-Uncertainty-Risk (C.U.R.): Here we propose a novel Fleet-DAgger algorithm. As the name suggests, C.U.R. does C-prioritization, followed by U-prioritization, followed by R-prioritization. R stands for Risk, which we define as the probability of constraint violation. Intuitively, idle robots should be reset in order to continue making progress, uncertain robots should receive more human supervision in areas with little to no reference behavior to imitate, and robots at risk should request human teleoperation to safety before an expensive hard reset. As in [247], we estimate the probability of constraint violation with a safety critic Q-function and initialize the safety critic on an offline dataset of constraint violations. C.U.R. also prioritizes differently at the beginning of execution for a parameterized length of time, during which constraint violations are assigned *zero priority* rather than high priority. Here, the intuition is that rather than attending to hard resets for an initially low-performing policy, human intervention should instead be spent on valuable teleoperation data that can improve the robot policy. Hence, during the initial period, constraint-violating robots remain idle and human attention is allocated to the teleoperation of a smaller number of robots.

5.5 Interactive Fleet Learning Benchmark

While many algorithms have been proposed for interactive learning [98, 171, 97, 276], to our knowledge there exists no unified benchmark for evaluating them. To facilitate reproducibility and standardized evaluation for IFL algorithms, we introduce the Interactive Fleet Learning Benchmark (IFLB). The IFLB is an open-source Python implementation of IFL with a suite of simulation environments and a modular software architecture for rapid prototyping and evaluation of new IFL algorithms.

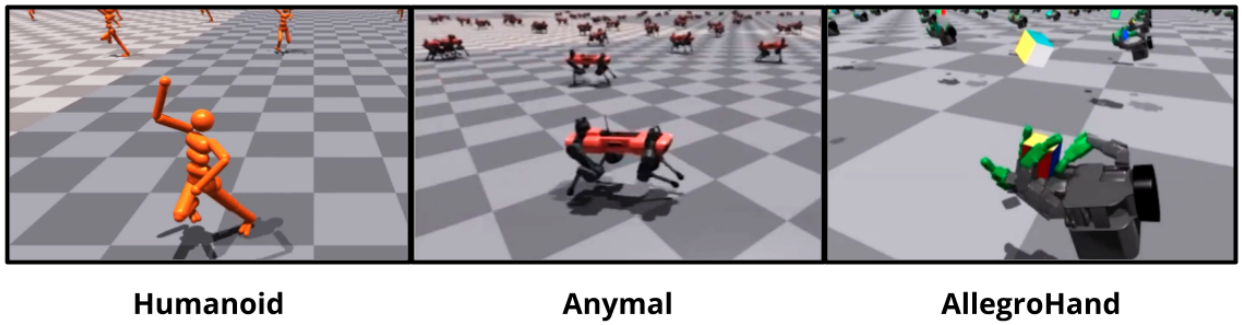


Figure 5.2: Isaac Gym benchmark environments in the IFLB.

Environments

The IFLB is built on top of NVIDIA Isaac Gym [157], a highly optimized software platform for end-to-end GPU-accelerated robot learning released in 2021, without which the simulation of hundreds of learning robots would be computationally intractable. The IFLB can run efficiently on a single GPU and currently supports the following 3 Isaac Gym environments with high-dimensional continuous state and action spaces (see Figure 5.2): (1) **Humanoid**, a bipedal legged locomotion task from OpenAI Gym [25], (2) **Anymal**, a quadruped legged locomotion task with the ANYmal robot by ANYbotics, and (3) **AllegroHand**, a task involving dexterous manipulation of a cube with a 4-finger Allegro Hand by Wonik Robotics. Constraint violation is defined as (1) the humanoid falling down, (2) the ANYmal falling down on its torso or knees, and (3) dropping the cube from the hand, respectively. End users can also add their own custom Isaac Gym environments with ease.

Software Architecture

The IFLB defines 3 interfaces for the development of IFL algorithms: (1) agents, (2) supervisors, and (3) allocations. An *agent* is an implementation of the robot fleet policy π_{θ_t} (Section 5.3), such as an IL or RL agent. A *supervisor* is an implementation of the supervisor policy π_H (Section 5.3), such as a fully trained RL agent, a model-based planner, or a teleoperation interface for remote human supervisors. Lastly, an *allocation* is an implementation of the priority function $\hat{p}$ (Section 5.4), such as C.U.R. priority or ThriftyDAgger priority. For reference, the IFLB includes an imitation learning agent, a fully trained RL supervisor using Isaac Gym's reference PPO [226] implementation, and all allocations from Section 5.4, which we use in our experiments. Users of the IFLB can flexibly implement their own IFL algorithms by defining new agents, supervisors, and allocations.

Given an agent, supervisor, allocation, and environment, the IFLB runs Fleet-DAgger as described in Section 5.4. IFLB allows flexible command line configuration of all parameters of the experiment (e.g., t_T, t_R, N, M) as well as the parameters of the agent, supervisor, and allocation. If desired, the code can also be modified to support families of IFL algorithms

other than Fleet-DAgger. The benchmark is available open-source at `https://github.com/BerkeleyAutomation/ifl_benchmark`.

5.6 Experiments

Metrics

Throughout online training, we measure four metrics at each timestep t: (1) the cumulative number of successful task completions across the fleet and up to time t; (2) cumulative hard resets (i.e., constraint violations); (3) cumulative idle time, i.e., how long robots spend idle in constraint-violating states waiting for hard resets; and (4) the return on human effort (ROHE, Equation 5.2), where reward is a sparse $r \in \{0, 1\}$ for successful task completion and cumulative human time is measured in hundreds of timesteps. For the Humanoid and Anymal locomotion environments, success is defined as reaching the episode horizon without constraint violation and with reward of at least 95% of that of the supervisor policy. For the goal-conditioned tasks, i.e., AllegroHand and the physical block-pushing task, success is defined by reaching the goal state.

IFLB Simulation Experiments

Experimental Setup: We evaluate all Fleet-DAgger algorithms in the 3 benchmark simulation environments: Humanoid, Anymal, and AllegroHand. We use reinforcement learning agents fully trained with PPO [226] as the algorithmic supervisor π_H. We initialize the robot policy π_{θ_0} with behavior cloning on an offline dataset of 5000 state-action pairs. For a fair comparison, the Constraint baseline is given additional offline data equal to the average amount of human time solicited by C.U.R. by operation time boundary T. The Random baseline's priority threshold is set such that in expectation, it reaches the average amount of human time solicited by C.U.R. by time T. Since Fleet-ThriftyDAgger requires a goal-conditioned task, it is only evaluated on AllegroHand. All training runs are executed with $N = 100$ robots, $M = 10$ humans, $t_T = 5$, $t_R = 5$, and operation time $T = 10000$, and are averaged over 3 random seeds. In the appendix, we provide additional experiments including ablation studies on each component of the C.U.R. algorithm and an analysis of hyperparameter sensitivity to the number of humans M, minimum intervention time t_T, and hard reset time t_R. The IFLB code provides instructions for reproducing results.

Results: We plot results in Figure 5.3. First, we observe that the choice of IFL algorithm has a significant impact on all metrics in all environments, indicating that allocation matters in the IFL setting. We also observe that the robot fleet achieves a higher throughput (number of cumulative task successes) with C.U.R. allocation than baselines in all environments at all times. C.U.R. also attains a higher ROHE, indicating more efficient use of human supervision. An increase in ROHE over time signifies that the improvement in the robot policy π_{θ_t} outpaces cumulative human supervision, indicating that the IFL algorithms learn

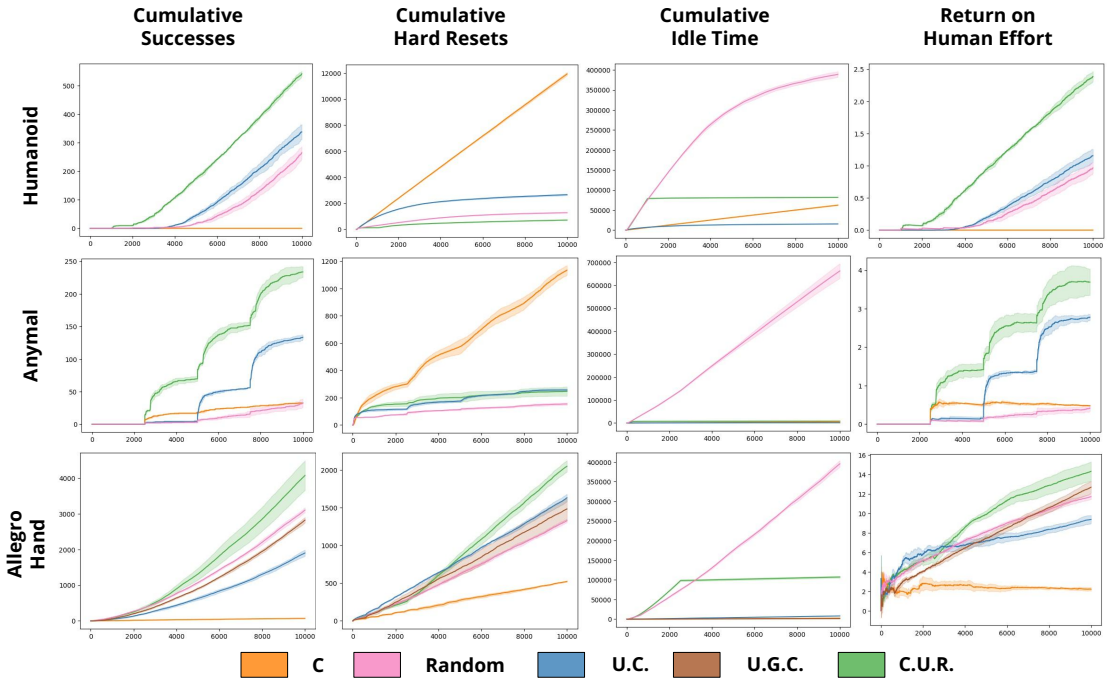


Figure 5.3: Simulation results in the IFLB with $N = 100$ robots and $M = 10$ human supervisors, where the x-axis is timesteps from 0 to $T = 10,000$. Shading indicates 1 standard deviation. The C.U.R. algorithm outperforms all baselines on all environments in terms of ROHE and cumulative successes. (Note that the shape of the Anymal curves is due to its success classification, episode horizon of 2500, and low hard resets.)

not only where to allocate humans but also when to *stop* requesting unnecessary supervision. C.U.R. also incurs fewer hard resets than baselines, especially Constraint, which must constantly hard reset robots with a low-performing offline policy. For AllegroHand, however, C.U.R. incurs higher hard resets and a smaller ROHE margin over baselines. We hypothesize that since the task is too challenging to execute without human supervision in the given fleet operation time, prioritizing hard resets ironically only gives the robots additional opportunities to violate constraints. We also see that C-prioritization effectively eliminates idle time; C.U.R. idle time flattens out after the initial period without C-prioritization.

Physical Block-Pushing Experiment

Experimental Setup: Finally, we evaluate Fleet-DAgger in a physical block-pushing experiment with $N = 4$ ABB YuMi robot arms and $M = 2$ human supervisors. Each robot arm has an identical setup for the block-pushing task that consists of a square wooden workspace, a small blue cube, and a cylindrical end-effector. See Figure 5.4 for the hardware setup. Constraint violation occurs when the cube hits the boundary or has moved

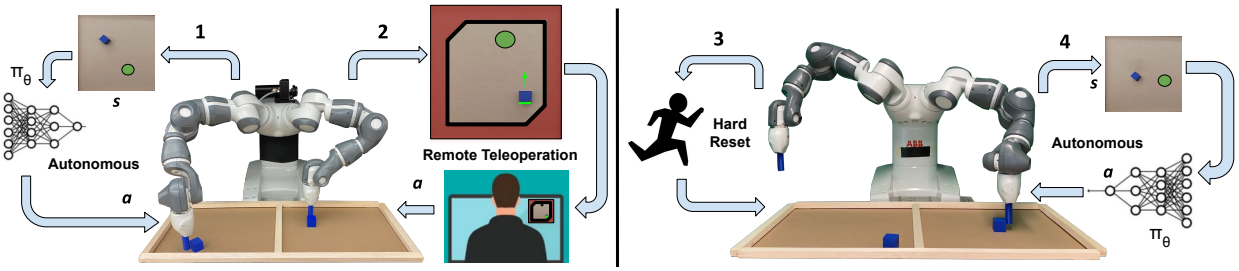


Figure 5.4: **Physical Task Setup:** an example timestep t in the physical experiment with 2 humans and 4 independent identical robot arms each executing the block pushing task. **Robot 1** queries robot policy π_{θ_t} for an action given an overhead image of the workspace and executes it in the workspace. **Robot 2** is teleoperated by a remote Human 1, where the human views the overhead image and specifies a pushing action through a user interface. The red region at the edges of the workspace are constraint violation regions. Human 2 is performing a physical hard reset for **Robot 3**, which has violated a constraint in a previous timestep. **Robot 4** autonomously executes the same robot policy as that of Robot 1 on its own state.

into regions out of reach for the end-effector at two opposite corners of the workspace. The objective of each robot is to reach a goal position randomly sampled from the allowable region of the workspace. At each timestep, the robot chooses one of four discrete pushing actions corresponding to pushing each of the four vertical faces of the cube orthogonally by a fixed distance. The robot policy takes an overhead image observation of the cube in the workspace with the goal programatically generated in the image. Hard resets are physical adjustments of the cube, while teleoperation is performed over the Internet by a remote human supervisor, who specifies one of the 4 pushing actions via a keyboard interface. We set $t_T = 3$, $t_R = 5$, and $T = 250$ for a total of 4×250 pushing actions per trial and run each algorithm with 3 random seeds. All algorithms are initialized with an offline dataset of 3750 image-action pairs (375 samples with $10\times$ data augmentation).

Results: We plot results in Figure 5.5. We observe that the C.U.R. algorithm achieves higher ROHE, higher cumulative successes, lower hard resets, and lower idle time than baselines, albeit by a small margin. Results suggest that (1) training an accurate safety critic is more difficult in high-dimensional image space, leading to a smaller gap between C.U.R. and U.C. (i.e., Fleet-EnsembleDAgger), and (2) U-prioritization in its current form is less suitable for real-world multimodal human supervisors than it is for deterministic algorithmic supervisors, resulting in a smaller increase in ROHE over time. Since a human may arbitrarily choose one of multiple equally suitable actions, high robot uncertainty over these actions does not necessarily translate to a need for human supervision.

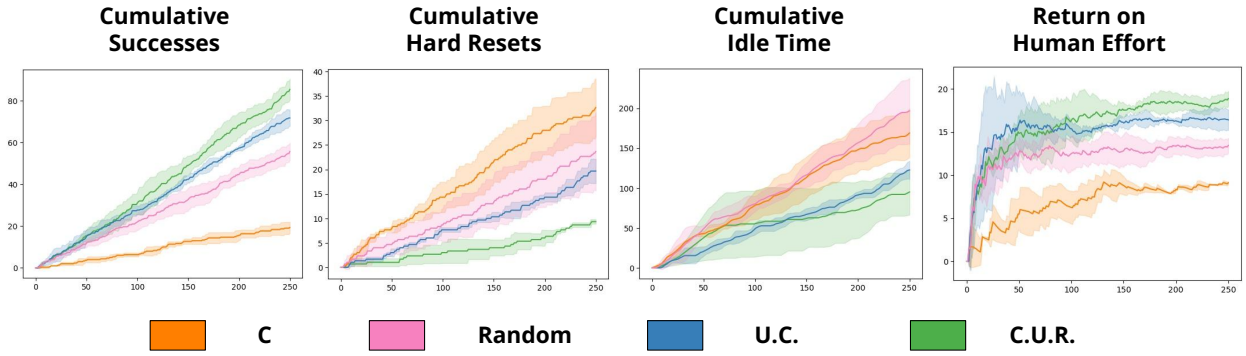


Figure 5.5: Physical results for the block-pushing task with 4 robots and 2 humans, where the x-axis is timesteps. C.U.R. achieves higher ROHE and cumulative successes as well as lower cumulative hard resets and idle time. Shading indicates 1 standard deviation.

5.7 Limitations and Future Work

The IFL formulation has a number of modeling assumptions that limit its generality. (1) The human supervisors are homogeneous, (2) all robots operate in the same state and action space, (3) all robots are independent and do not coordinate with each other, (4) humans have perfect situational awareness [46] and can move to different robots without any switching latency, (5) hard reset time is constant, and (6) timesteps are synchronous without network latency or other communication issues [131]. In terms of experiments, the simulations have algorithmic rather than human supervision, and the physical task is relatively straightforward with discrete planar actions.

Future work involves lifting the assumptions above. In the next chapter, we introduce an algorithm that lifts the first assumption by enabling learning from nonhomogeneous supervisors. Other interesting directions are reinforcement learning algorithms for IFL and more large-scale physical experiments. We hope that other robotics researchers will develop their own IFL algorithms and evaluate them using the benchmark toolkit to accelerate progress.

Chapter 6

IIFL: Implicit Interactive Fleet Learning

In this chapter, we extend implicit behavioral cloning [75] to the IFL setting and introduce a novel technique for estimating uncertainty in energy-based models [141]. This enables learning from multimodal data and heterogeneous human behavior.

6.1 Introduction

Imitation learning (IL), the paradigm of learning from human demonstrations and feedback, has been applied to diverse tasks such as autonomous driving [191, 204, 39], robot-assisted surgery [193, 129], and deformable object manipulation [227, 14, 101]. The most common IL algorithm is behavior cloning (BC) [204], where the robot policy is derived via supervised machine learning on an offline set of human task demonstrations. Since BC can suffer from distribution shift between the states visited by the human and those visited by the robot, interactive IL (IIL) algorithms including DAgger [215] and variants [97, 125, 171] iteratively improve the robot policy with corrective human interventions during robot task execution. These algorithms are typically designed for the single-robot, single-human setting; *interactive fleet learning* (IFL) [99] extends IIL to multiple robots and multiple human supervisors. However, learning from multiple humans can be unreliable as the data is often multimodal.

Training data is *multimodal* when the same state is paired with multiple (correct) action labels: $\{(s, a_i), (s, a_j), \dots\}, a_i \neq a_j$. Almost all robot tasks such as grasping, navigation, motion planning, and manipulation can be performed in multiple equally correct ways; as a result, almost all demonstration data has some degree of multimodality. Multimodality is especially severe when learning from different human supervisors with varying preferences and proficiency, as they demonstrate the same task in different ways [162]. Multimodality can also occur in the demonstrations of one individual human who may make mistakes, become more proficient at the task over time, or execute a different valid action when subsequently encountering the same state [162, 183].

Florence et al. [75] propose Implicit Behavior Cloning (IBC), an IL algorithm that trains an energy-based model (EBM) [141] to represent state-action mappings *implicitly* rather than explicitly. While this makes model training and inference more computationally expensive (Section 6.6), implicit models can represent multiple actions for each state. This property allows them to handle both single-human multimodality and multi-human heterogeneity, as they are indistinguishable from a data-centric perspective. However, IBC suffers from the same distribution shift as (Explicit) BC.

In this paper we combine implicit models with interactive fleet learning to facilitate interactive learning from multiple humans. See Figure 6.1 for intuition. As existing IFL algorithms rely on estimates of epistemic uncertainty like the output variance among an ensemble of networks, which are incompatible with implicit models (Section 6.4), we propose a new technique for estimating the epistemic uncertainty in EBMs using Jeffreys divergence [111].

This paper makes the following contributions: (1) Implicit Interactive Fleet Learning (IIFL), the first IIL algorithm to use implicit policies, (2) a novel metric for estimating uncertainty in energy-based models, (3) simulation experiments with a fleet of 100 robots and 10 heterogeneous algorithmic supervisors, (4) physical experiments with a fleet of 4 robots and 2 heterogeneous human supervisors. Open-source Python code is available at `https://github.com/BerkeleyAutomation/IIFL`.

6.2 Preliminaries and Related Work

Interactive Imitation Learning

Learning from an offline set of human task demonstrations with behavior cloning (i.e., supervised learning) is an intuitive and effective way to train a robot control policy [12, 204, 227, 191]. However, behavior cloning can suffer from distribution shift [215], as compounding approximation errors and real-world data distributions (e.g., variable lighting in a warehouse) can lead the robot to visit states that were not visited by the human. To mitigate distribution shift, Ross et al. [215] propose dataset aggregation (DAgger), an IIL algorithm which collects online action labels on states visited by the robot during task execution and iteratively improves the robot policy. Since DAgger can request excessive queries to a human supervisor, several IIL algorithms seek to reduce human burden by intermittently ceding control to the human during robot execution based on some switching criteria [125, 97, 276]. Human-gated IIL [125, 241, 148] has the human decide when to take and cede control, while robot-gated IIL [98, 97, 171, 276] has the robot autonomously decide. Hoque et al. [99] propose Interactive Fleet Learning (IFL), which generalizes robot-gated IIL to multiple robots supervised by multiple humans. In this work, we consider the IFL setting.

Sun et al. [243] propose a method for interactive imitation learning from heterogeneous experts, but their method is not based on implicit policies and is limited to autonomous driving applications. Gandhi et al. [81] also interactively learn from multiple experts and

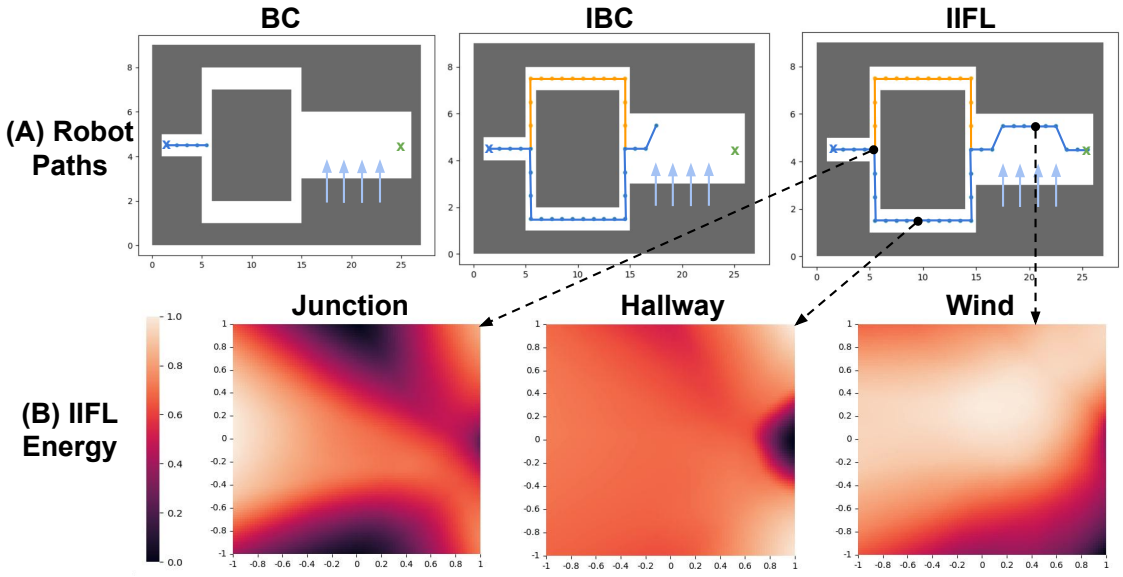


Figure 6.1: In the 2D navigation experiments from Section 6.5, the robot must navigate from the blue X marker on the left to the green X marker on the right, where the robot can go either above or below the rectangular grey obstacle and continue through a section subject to upward wind forces (blue arrows) that shift commanded motions upward. **(A) Robot Trajectories:** After training on 100 demonstrations of the two paths around the obstacle, pure behavior cloning cannot make progress past the fork due to multimodal demonstrations, while Implicit Behavior Cloning cannot overcome the distribution shift due to wind in the $+y$ direction at execution time (denoted in light blue). IIFL reaches the goal by handling both multimodality and distribution shift. **(B) Implicit Interactive Fleet Learning Energy:** We display normalized IIFL energy distributions from representative states in the trajectory. Lower energy (darker) indicates a more optimal action, and the x and y axes are the 2D action deltas $\hat{a}$ that the robot can execute (which can be mapped directly onto the corresponding 1×1 cell in the maze). At the junction point, both upward and downward actions attain low energy; in a straight hallway, the rightmost actions attain low energy; in the windy area, actions toward the lower right corner (making progress toward the goal while fighting the wind) attain low energy.

propose actively soliciting the human supervisors to provide demonstrations that are compatible with the current data. However, this prevents the robot from learning alternative modes and requires the human supervisors to comply with suggestions, which may not occur due to human suboptimality, fatigue, or obstinacy [47].

Robot Learning from Multimodal Data

Learning from multimodal demonstrations is an active challenge in machine learning and robotics. A mixture density network [20] is a popular approach that fits a (typically Gaussian) mixture model to the data, but it requires setting a parameter for how many modes to fit, which may not be known a priori. When actions can be represented as pixels in an image (e.g., pick points), a Fully Convolutional Network [233] can be applied to learning pixelwise multimodality [101, 275]. Shafiullah et al. [231] propose Behavior Transformers, a technique that applies the multi-token prediction of Transformer neural networks [258] to imitation learning. Other Transformer-based policies report similar benefits for multimodal data [237, 113]; however, these approaches require action discretization to cast behavior prediction as next-token prediction. In a very recent paper, Chi et al. [43] introduce diffusion policies, an application of diffusion models [96] to imitation learning from multimodal data.

Florence et al. [75] propose implicit behavior cloning, a technique that trains a conditional energy-based model [141] and is found to outperform (explicit) BC and mixture density networks in their experiments. As opposed to explicit models that take the form $\pi : \mathcal{S} \rightarrow \mathcal{A}$, implicit models take the form of a scalar-valued function $E : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$; the action is an input rather than an output of the model. To sample an action from the policy, instead of evaluating the explicit model $\hat{a} = \pi(s)$, the implicit model must perform optimization over E conditioned on state s:

$$\hat{a} = \arg\min_{a \in \mathcal{A}} E(s, a) \tag{6.1}$$

In this work, we combine IBC with IFL to mitigate the effects of both distribution shift and multimodality. To our knowledge, we are the first to extend implicit policies to interactive IL.

Jeffreys Divergence

The Jeffreys divergence [111] is a statistical measure of the distance between two probability distributions and is a symmetric version of the Kullback-Leibler (KL) divergence:

$$D_J(P\|Q) = D_{KL}(P\|Q) + D_{KL}(Q\|P).$$

The KL divergence is widely used in machine learning algorithms, most commonly in variational autoencoders [130] and generative adversarial networks [85]. It has also been used for dimensionality reduction [153], information bottlenecks [249], and policy gradient methods for reinforcement learning [225, 226]. The Jensen-Shannon divergence [145] is another symmetric KL divergence that sums the KL divergences of both distributions against the

mixture of the two, but neither the Jensen-Shannon nor the asymmetric KL divergences have the structural properties that make Jeffreys divergence amenable to our setting (Section 6.4). Nielsen [182] derives a proposition similar to Identity 1 (Section 6.4) with Jeffreys divergence for exponential families but does not apply it to energy-based models. To our knowledge, IIFL is the first algorithm to use Jeffreys divergence for uncertainty estimation in energy-based models, exploiting its structural properties for fast computation.

6.3 Problem Statement

We consider the interactive fleet learning (IFL) setting proposed by Hoque et al. [99]. A fleet of N robots operate in parallel independent Markov Decision Processes (MDPs) that are identical apart from their initial state distributions. The robots can query a set of $M < N$ human supervisors with action space $\mathcal{A}_H = \mathcal{A} \cup \{R\}$, where $a \in \mathcal{A}$ is teleoperation in the action space of the robots and R is a "hard reset" that physically resets a robot in a failure state (e.g., a delivery robot tipped over on its side). As in [99], we assume that (1) the robots share policy $\pi_{\theta_t} : \mathcal{S} \to \mathcal{A}$, (2) the MDP timesteps are synchronous across robots, and (3) each human can only help one robot at a time. However, unlike the original IFL formulation [99], we do *not* assume that the human supervisors are homogeneous; instead, each human i may have a unique policy $\pi_H^i : \mathcal{S} \to \mathcal{A}_H$. Furthermore, each π_H^i may itself be nondeterministic and multimodal, but is assumed to be optimal or nearly optimal.

An IFL supervisor allocation algorithm is a policy ω that determines the assignment $\boldsymbol{\alpha}^t$ of humans to robots at time t, with no more than one human per robot and one robot per human at a time:

$$\omega : (\mathbf{s}^t, \pi_{\theta_t}, \cdot) \mapsto \boldsymbol{\alpha}^t \in \{0,1\}^{N \times M} \quad \text{s.t.} \quad \sum_{j=1}^{M} \boldsymbol{\alpha}_{ij}^t \leq 1 \text{ and } \sum_{i=1}^{N} \boldsymbol{\alpha}_{ij}^t \leq 1 \quad \forall i, j. \tag{6.2}$$

The allocation policy ω in IFL must be autonomously determined with robot-gated criteria [97, 171] rather than human-gated criteria [125, 241, 148] in order to scale to large ratios of N to M. The IFL objective is to find an ω that maximizes return on human effort (ROHE), defined as the average performance of the robot fleet normalized by the amount of human effort required [99]:

$$\max_{\omega \in \Omega} \mathbb{E}_{\tau \sim p_{\omega,\theta_0}(\tau)} \left[\frac{M}{N} \cdot \frac{\sum_{t=0}^{T} \bar{r}(\mathbf{s}^t, \mathbf{a}^t)}{1 + \sum_{t=0}^{T} \|\omega(\mathbf{s}^t, \pi_{\theta_t}, \boldsymbol{\alpha}^{t-1}, \mathbf{x}^t)\|_F^2} \right] \tag{6.3}$$

where $\| \cdot \|_F$ is the Frobenius norm, T is the amount of time the fleet operates (rather than an individual episode horizon), and θ_0 are the initial parameters of π_{θ_t}.

6.4 Approach

Preliminaries: Implicit Models

We build on Implicit Behavior Cloning [75]. IBC seeks to learn a conditional energy-based model $E : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $E(s, a)$ is the scalar "energy" for action a conditioned on state s. Lower energy indicates a higher correspondence between s and a. The energy function defines a multimodal probability distribution π of action a conditioned on state s:

$$\pi(a|s) = \frac{e^{-E(s,a)}}{Z(s)} \tag{6.4}$$

where $Z(s)$ is a normalization factor known as the "partition function." In practice, we estimate E with a learned neural network function approximator E_θ parameterized by θ and train E_θ on samples $\{s_i, a_i\}$ collected from the expert policies π_H. Training E_θ is described in Appendix D.2.

Implicit Interactive Dataset Aggregation

Behavior cloning is prone to distribution shift due to compounding approximation errors [215], and any data-driven robot policy may encounter edge cases during execution that are not represented in the training data [99]. We extend IBC to interactive imitation learning using dataset aggregation of online human data, and iteratively update the shared robot policy with the aggregate dataset at a fixed interval $1 \leq \hat{t} \leq T$ via supervised learning, as in DAgger [215] and variants [125, 99]:

$$\begin{cases} D^{t+1} \leftarrow D^t \cup D_H^t, \text{ where } D_H^t := \{(s_i^t, \pi_H^j(s_i^t)) : \pi_H^j(s_i^t) \neq R \text{ and } \sum_{j=1}^M \boldsymbol{\alpha}_{ij}^t = 1\} \\ \pi_{\theta_t} \leftarrow \arg\min_\theta \mathcal{L}(\pi_\theta, D^t), \texttt{ if } t \equiv 0 \pmod{\hat{t}} \end{cases}$$

where $\pi_H^j(s_i^t)$ is the teleoperation action from human j for robot i at time t, and $\boldsymbol{\alpha}_{ij}^t$ is the assignment of human j to robot i at time t, as in Equation 6.2. Ross et al. [215] show that such a policy incurs approximation error that is linear in the time horizon rather than quadratic, as in behavior cloning.

Uncertainty Estimation for EBMs

While prior work computes the output variance among a bootstrapped ensemble of neural networks to estimate epistemic uncertainty [51, 171, 97], this approach is not applicable to implicit policies because multimodality results in a false positive: different ensemble members may select equally optimal actions from different modes, resulting in high variance despite high certainty. Furthermore, training and inference in EBMs are much more computationally expensive than in explicit models (Section 6.6), making ensembles of 5+ models impractical.

Finally, inference in implicit models is nondeterministic, creating an additional source of variance that is not due to uncertainty.

The notion of ensemble disagreement can still be applied to EBMs by considering the action *distributions* at a given state rather than the single predicted actions. At states within the distribution of the human data, a bootstrapped EBM will predict action distributions that are concentrated around the human actions. However, outside of the human data distribution, the models have no reference behavior to imitate, and will likely predict different conditional action distributions due to random initialization, stochastic optimization, and bootstrapping. Accordingly, we propose bootstrapping 2 implicit policies and calculating the Jeffreys divergence D_J [111] between them as a measure of how their conditional action distributions differ at a given state. Jeffreys divergence in this setting has two key properties: (1) it is symmetric, which is useful as neither bootstrapped policy is more correct than the other, and (2) it is computationally tractable for EBMs as it does not require estimating the partition function $Z(s)$ (Equation 6.4). To show (2), we derive the following novel identity (proof in Appendix D.1):

Identity 1. *Let E_1 and E_2 be two energy-based models that respectively define distributions π_1 and π_2 according to Equation 6.4. Then,*

$$D_J\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) = \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right].$$

Crucially, the intractable partition functions do not appear in the expression due to the symmetry of Jeffreys divergence. We estimate the expectations in Identity 1 using Langevin sampling. Note that this method is not limited to the interactive IL setting and may have broad applications for any algorithms or systems that use energy-based models. We provide more intuition on the proposed metric in Figure D.1 in the appendix and consider how this method may be generalized to a greater number of models in Appendix D.3.

Energy-Based Allocation

To extend IBC to the IFL setting, we synthesize the Jeffreys uncertainty estimate with Fleet-DAgger [99]. Specifically, we set the Fleet-DAgger priority function $\hat{p} : (s, \pi_{\theta_t}) \to [0, \infty)$ to prioritize robots with high uncertainty, followed by robots that require a hard reset R. This produces a supervisor allocation policy ω with Fleet-EnsembleDAgger, the U.C. (Uncertainty-Constraint) allocation scheme in [99]. We refer to the composite approach as IIFL.

6.5 Experiments

Simulation Experiments: 2D Navigation

To evaluate the correctness of our implementation and provide visual intuition, we first run experiments in a 2D pointbot navigation environment. See Figure 6.1 for the maze

environment, representative trajectories, and energy distribution plots. We consider discrete 2D states $s = (x, y) \in \mathbb{N}^2$ (the Cartesian pose of the robot) and continuous 2D actions $a = (\Delta x, \Delta y) \in [-1, 1]^2$ (relative changes in Cartesian pose). The maze has a fixed start and goal location and consists of a forked path around a large obstacle followed by a long corridor. An algorithmic supervisor provides 100 demonstrations of the task, randomly choosing to go upward or downward at the fork with equal probability. Since a model can simply overfit to the demonstrations in this low-dimensional environment, to induce distribution shift we add "wind" at execution time to a segment of the right corridor with magnitude 0.75 in the $+y$ direction.

In 100 trials, (explicit) BC achieves a 0% success rate, IBC achieves a 0% success rate, and IIFL achieves a 100% autonomous success rate (i.e., robot-only trajectories without human interventions, after interactive training). In Figure 6.1 we observe that BC cannot pass the fork due to averaging the two modes to zero. Meanwhile, IBC is not robust to the distribution shift: once the wind pushes the robot to the top of the corridor, it does not know how to return to the center. We also observe that the IIFL energy distributions in Figure 6.1(B) reflect the desired behavior in accordance with intuition.

Simulation Experiments: IFL Benchmark

Environments: Evaluating IIFL in simulation is uniquely challenging as it requires all of the following, precluding the use of most existing benchmarks in similar papers: (1) efficient simulation of large robot fleets, (2) simulation of multiple algorithmic humans, (3) interactive human control, and (4) heterogeneous human control, which is difficult to specify in joint space. To accommodate these requirements, following prior work [99] we evaluate with Isaac Gym [157] and the IFL Benchmark [99]. We separate these experiments into two domains: (1) homogeneous human control in 3 environments (Ball Balance, Ant, Anymal) to compare with prior IFL algorithms that assume unimodal supervision; (2) heterogeneous human control in FrankaCubeStack, the only Isaac Gym environment with Cartesian space control. More details are available in Appendix D.4.

Metrics: Following prior work [99], we measure the total successful task completions across the fleet and the total number of hard resets. For interactive algorithms, we also measure the return on human effort (Equation 6.3) where reward is a sparse $r \in \{0, 1\}$ for task completion. Task execution is deemed successful if the robot completes its trajectory without a hard reset and reaches 95% of expert human reward.

Baselines: We compare IIFL to the following baselines: (explicit) BC, IBC, (explicit) IFL (specifically, Fleet-EnsembleDAgger [99]), and IIFL-Random (IIFL-R), which is an ablation of IIFL that allocates humans to robots randomly instead of using the Jeffreys uncertainty estimate. Human supervisors for BC and IBC perform only hard resets (i.e., no teleoperation) during execution.

Experimental Setup: We run experiments with a fleet of $N = 100$ robots and $M = 10$ algorithmic supervisors, where the supervisors are reinforcement learning agents trained with Isaac Gym's reference implementation of PPO [226]. All training runs have hard reset time

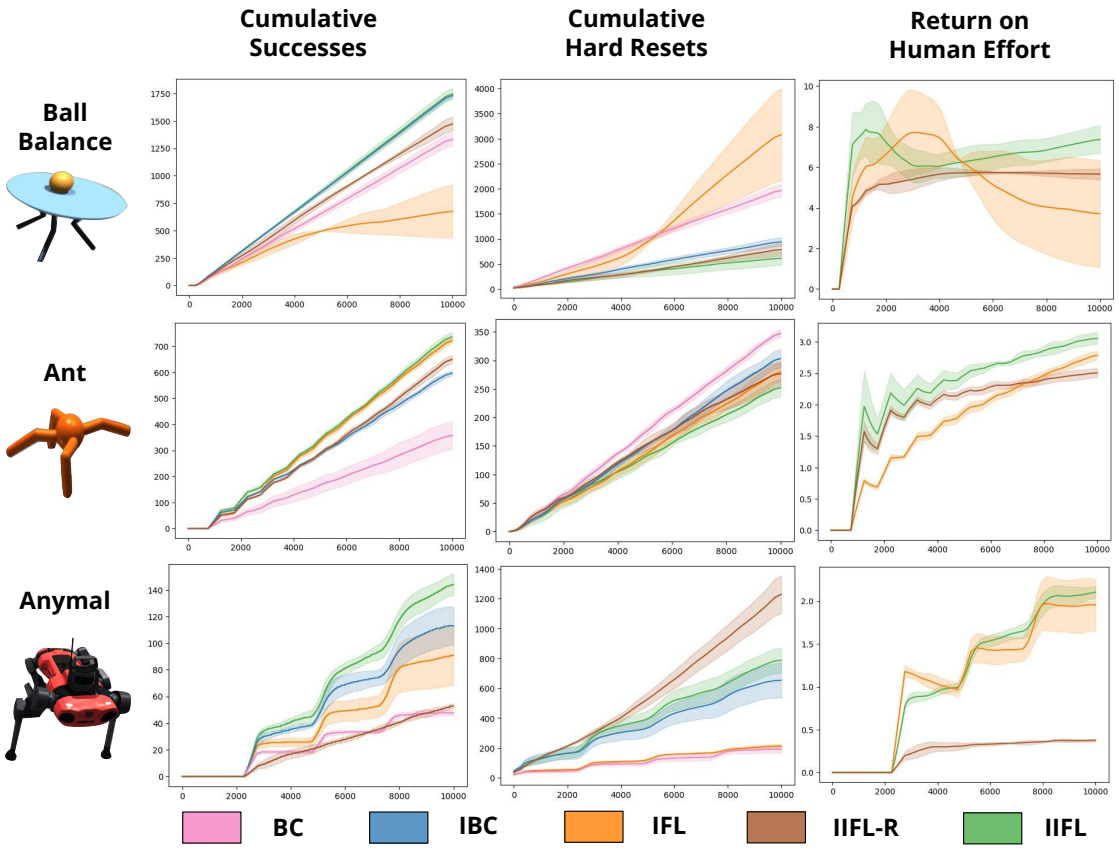


Figure 6.2: IFL Benchmark simulation experiment results. Despite unimodal supervision, IIFL is competitive with or outperforms IFL and other baselines across 3 environments, suggesting benefits of implicit policies beyond robustness to multimodality. Shading represents ± 1 standard deviation.

Algorithm	Avg. Reward	Task Successes	ROHE
BC	29.27 ± 14.05	0.3 ± 0.5	N/A
IBC	24.96 ± 0.83	0.0 ± 0.0	N/A
IFL	230.39 ± 53.41	7.0 ± 2.2	2.30 ± 0.53
IIFL-R	166.24 ± 28.63	0.0 ± 0.0	1.66 ± 0.29
IIFL	$\mathbf{784.26 \pm 122.41}$	$\mathbf{26.7 \pm 4.5}$	$\mathbf{7.84 \pm 1.22}$

Table 6.1: Execution results from the FrankaCubeStack Isaac Gym environment with 4 heterogeneous expert policies. IIFL significantly outperforms the baselines in average reward, task successes, and return on human effort.

$t_R = 5$ timesteps, minimum intervention time $t_T = 5$ timesteps, and fleet operation time $T = 10,000$ timesteps [99], and are averaged over 3 random seeds. The initial robot policy π_{θ_0} for all algorithms is initialized with behavior cloning on 10 full task demonstrations. While IFL trains at every timestep following prior work [99], the implicit interactive algorithms train at intervals of 1000 timesteps with an equivalent total amount of gradient steps for increased stability of EBM training.

FrankaCubeStack, in which a Franka arm grasps a cube and stacks it on another (see Appendix D.4 for images and details), has several differences from the other 3 environments. First, since it allows Cartesian space control, we can script 4 *heterogeneous* supervisor policies with grasps corresponding to each face of the cube; the $M = 10$ supervisors are split into 4 groups, each of which has a unique policy. Second, due to the difficulty of scripting interactive experts, the online interventions take place at execution-time (i.e., the robot policy is frozen). Third, since there is no notion of catastrophic failure in the cube stacking environment, we do not report hard resets as there are none.

Results: The results are shown in Figure 6.2 and Table 6.1. In the homogeneous control experiments, we observe that IIFL rivals or outperforms all baselines across all metrics, with the exception of hard resets in the Anymal environment. We hypothesize that the latter results from learning more "aggressive" locomotion that makes greater progress on average but is more prone to failure. These results suggest that implicit policies may have desirable properties over explicit policies such as improved data efficiency and generalization even when multimodality is *not* present in the data, as suggested by prior work [75]. The severity of distribution shift due to compounding approximation error [215] in the homogeneous experiments roughly corresponds to the performance gap between BC and IFL (or IBC and IIFL). Surprisingly, (explicit) IFL underperforms BC in Ball Balance; we hypothesize that this is due to its frequent policy updates on a shifting low-dimensional data distribution. In the FrankaCubeStack environment, IIFL significantly outperforms the baselines across all metrics, indicating the value of implicit policies for heterogeneous supervision. The 74% performance gap between IFL and IIFL corresponds to the severity of multimodality in this environment. Only IFL and IIFL attain nontrivial success rates; while IIFL-R makes progress, it is not able to successfully stack the cube, suggesting that IIFL allocates human attention more judiciously.

Physical Experiments: Pushing Block to Target Point amid Obstacle

Experimental Setup: To evaluate IIFL in experiments with real-world human multimodality and high-dimensional state spaces, we run an image-based block-pushing task with a fleet of $N = 4$ ABB YuMi robot arms operating simultaneously and $M = 2$ human supervisors, similar to Hoque et al. [99]. See Figure 6.3 for the physical setup. Each robot has an identical square workspace with a small blue cube and rectangular pusher as an end effector. Unlike Hoque et al. [99], we add a square obstacle to the center of each workspace. The task for each

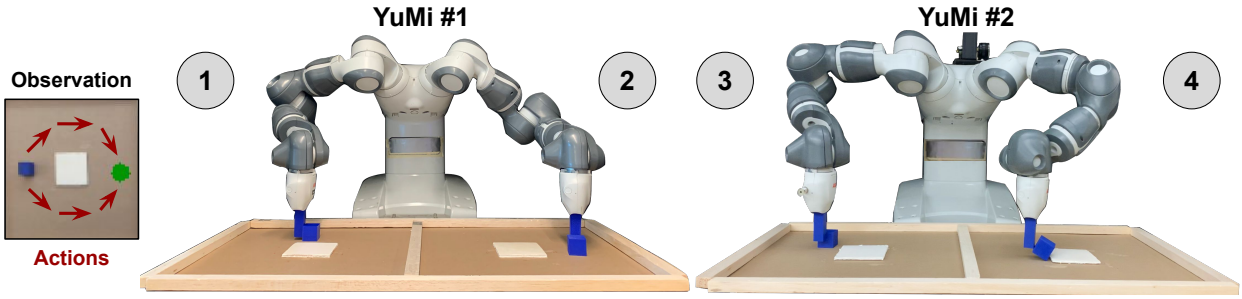


Figure 6.3: Physical experiment setup with 2 ABB YuMi robots for a total of 4 independent arms.

robot is to push the cube to a goal region diametrically opposite the cube's initial position without colliding with the walls or the obstacle. Once this is achieved, the goal region is procedurally reset based on the new cube position. As described in Section 6.3, the role of human superivsion is to (1) teleoperate when requested and (2) provide a physical hard reset when requested. When both paths to the goal are equidistant, Human 1 pushes the cube *clockwise* around the obstacle while Human 2 pushes the cube *counterclockwise*; if one path is closer, the human takes that path. Hard resets R are defined to be collisions of the cube with the obstacle or the boundaries of the workspace. Furthermore, unlike the discrete action space in Hoque et al. [99], we use a continuous 2D action space of $a = (\Delta x, \Delta y)$ that corresponds to the vector along which to push the block, starting from the block's center. We run 3 trials of each algorithm in Table 6.2 for $T = 150$ timesteps; see Appendix D.4 for more details.

Results: The results are shown in Table 6.2. We observe that implicit policies are crucial for success, as the explicit methods rarely reach the goal and incur many hard resets. Results suggest that IIFL improves the success rate by 80% over IBC and improves ROHE by 4.5× over IFL. However, IIFL incurs a similar number of hard resets to IBC. We hypothesize that the duration of the physical experiment, difficult to extend due to the significant robot and human time required, is insufficient to learn subtle collision avoidance behaviors that noticeably reduce the number of hard resets.

6.6 Limitations and Future Work

Since IIFL extends IBC, it inherits some of its limitations. First, model training and inference require 18× and 82× more computation time than explicit methods: on one NVIDIA V100 GPU, we measure implicit training to take an additional 0.34 seconds per gradient step and implicit inference to take an additional 0.49 seconds. Second, Florence et al. [75] find that IBC performance falls on some tasks when the action space dimensionality is very high ($|\mathcal{A}| > 16$); we do not observe this in our experiments as $|\mathcal{A}| \leq 12$ but IIFL likely

Algorithm	Successes (↑)	Hard Resets (↓)	ROHE (↑)
BC	2.0 ± 0.8	51.0 ± 0.8	N/A
IBC	20.3 ± 4.1	$\mathbf{35.3 \pm 6.8}$	N/A
IFL	7.0 ± 0.8	47.3 ± 0.5	0.13 ± 0.01
IIFL	$\mathbf{36.3 \pm 1.2}$	37.0 ± 2.2	$\mathbf{0.71 \pm 0.01}$

Table 6.2: Physical block pushing experiment results. IIFL outperforms all baselines in number of task successes and ROHE and explicit methods in hard resets. Implicit BC and IIFL incur similar amounts of hard resets.

incurs this property with higher-dimensional actions. Third, while it is 7× faster than alternate methods for implicit models and has sub-second latency for a fleet of 100 robots, IIFL uncertainty estimation is nevertheless 340× slower than its highly efficient explicit counterpart (Appendix D.4). Finally, the real-world evaluation of IIFL is limited to block pushing with fixed block properties; more comprehensive evaluation of IIFL in a wider range of physical domains is required to assess its full applicability.

Future work involves evaluating IIFL in additional physical environments as well as extending recently proposed alternative approaches for handling multimodality such as Behavior Transformers [231] and Diffusion Policies [43] to the IFL setting. It will also be important to develop algorithms that effectively learn from human demonstrations that are not only multimodal but also suboptimal. We note that as the Jeffreys uncertainty quantification method does not rely on any IFL assumptions, it may be broadly useful beyond this setting to any applications involving Boltzmann distributions and EBMs.

Part III

Systems for Remote Fleet Supervision

Chapter 7

Real-Time Remote Robot Manipulation

Here we study a prototype robot workcell that enables research and evaluation in real time remotely over the Internet. We conduct a case study of remote robotics research with the first systematic benchmarking of fabric manipulation algorithms on physical hardware.

7.1 Introduction

Reproducibility is the cornerstone of scientific progress. It allows researchers to verify results, assess the state of the art, and build on prior work. Recent advances in computer vision (CV), for instance, were facilitated by the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [66], a standard benchmark in CV literature.

In robotics, there is no equivalent benchmark. Simulation benchmarks [238, 25, 146] are useful but cannot replace physical experiments as the "reality gap" remains prohibitively large [55]. Physical robots are expensive and vary greatly in their capabilities and morphologies. Each research lab has a unique hardware setup, making it difficult to reliably compare results. Cost also poses a significant obstacle to individuals or institutions who wish to perform robotics research, but lack the resources to do so.

One option is to provide shared access to a remote hardware testbed via the Cloud. In this paper, we describe algorithms and experiments performed entirely remotely using *Reach*, a prototype hardware testbed from Robotics at Google [266]. Reach includes several physical robot workcells and open source software for remote execution of control policies in real time. Each workcell is configured for a benchmark task: one such task is folding a T-shirt with a UR5 robot arm and 3-jaw piSOFTGRIP gripper [201] (Figure 7.1).

While folding T-shirts and other garments is a ubiquitous daily task for humans, manipulating fabric remains challenging for robots. Fabric is difficult to model due to its infinite-dimensional state space, complex dynamics, and high degree of self-occlusion. Furthermore, accurately simulating gripper contact mechanics and fabric self-collision remains

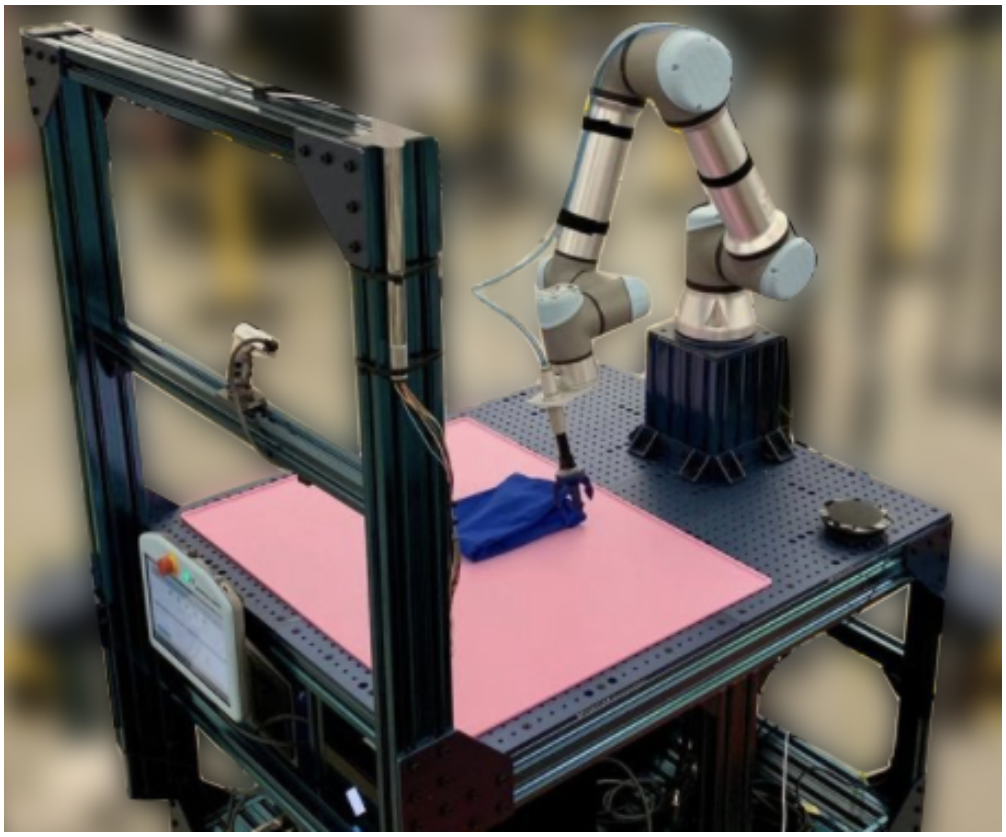


Figure 7.1: Reach cloud robotics workcell developed by Robotics at Google.

elusive for existing fabric simulators due to challenges in modeling deformation, friction, and electrostatic forces [146, 227], highlighting the need for physical benchmarking.

This paper makes the following contributions: (1) four novel learning-based algorithms for the unimanual folding task, (2) the first physical benchmarking of fabric manipulation algorithms, and (3) a case study of robotics research performed exclusively using a remotely managed robot workcell. This paper does *not* contribute the design of the Reach platform, which is being developed by a larger team at Google [266].

7.2 Related Work

Remote Testbeds: Remote robotics testbeds include Robotarium [202] for swarm robotics and Duckietown [195] for autonomous driving. The most similar remote testbed is from Bauer et al. [17], who hosted the online "Real Robot Challenge" for manipulation in 2020 and 2021 at Neural Information Processing Systems (NeurIPS). Six robotics groups from around the world were able to access their tri-finger robot [269] remotely via the Internet and evaluate their algorithms on the shared infrastructure. Our study differs from this

project in the following ways: (1) they consider dexterous manipulation of rigid objects while we consider deformable object manipulation; (2) they use a custom tri-finger robotic system while we use a UR5 robot arm, standard in industrial settings; and (3) the Real Robot Challenge submissions are either learning-free [79, 35] or learned only in simulation [168, 5], while we consider learning algorithms trained on real data.

Reproducibility in Robotics: Several other approaches have been proposed for facilitating reproducibility in robotics research. One direction is benchmarking in simulation, where evaluation is inexpensive and reproducible. Simulation environments have been developed for robot locomotion [25], household tasks [238], and deformable object manipulation [146]. While researchers have made significant progress on these benchmarks, especially using reinforcement learning [89], such advances do not readily transfer to physical robots [55]. Another initiative for improving reproducibility is development of a low-cost open source platform that can be assembled independently by different labs [273, 269]. A third approach considers benchmarking performance on large offline datasets such as robot grasps on 3D object models, e.g., EGAD [174] and Dex-Net [155]; RGBD scans and meshes of real-world common household objects, e.g., the YCB Object and Model set [33]; and video frames of robot experience, e.g., RoboNet [64]. These datasets have been used to explore and compare algorithms [128], but they limit evaluation to states within the dataset.

Autonomous Fabric Folding: Autonomous fabric manipulation is an active challenge in robotics. Maitin-Shepard et al. [156] and Doumanoglou et al. [68] present early approaches to reliably fold towels and garments, respectively, from crumpled initial configurations. Weng et al. [264] and Ha et al. [88] develop learning-based algorithms for fabric manipulation using optical flow and dynamic flinging motions respectively. However, these approaches were evaluated on dual-armed robots, which require coordination and are more costly. There has been recent interest in learning algorithms for unimanual (single-arm) fabric manipulation [100, 227]. These achieve strong results on fabric smoothing and folding tasks, but robust and precise unimanual T-shirt folding remains an open problem. Lin et al. [146] propose an environment for fabric manipulation and benchmark several learning algorithms, but limit results to simulation. Garcia-Camacho et al. [82] propose benchmarking tasks for bimanual fabric manipulation but allow robot hardware to vary and do not evaluate learning algorithms.

7.3 The Google Reach Testbed

In this section, we review the details of the Google Cloud Robotics testbed [266] that are most salient for this case study.

Hardware

See Figure 7.1 for an image of the workcell. The robot is a single Universal Robot UR5e arm equipped with a Piab piSOFTGRIP vacuum-driven soft 3-jaw gripper [201]. The workcell is

equipped with 4 Intel Realsense D415 cameras which each capture 640 × 360 RGB images at 20 FPS and 640 × 360 depth images at 1 FPS. The worksurface is a bright pink silicone mat and the garment is a blue crew-neck short sleeve T-shirt. The workcell is maintained by lab technicians who are onsite 8 hours a day to reset the robot and troubleshoot system-level errors.

Software

Reach includes PyReach, an open source Python library developed by Robotics at Google for interfacing with the Reach system. The software includes infrastructure for authenticated users to establish a network connection with the robot server over the Internet, a viewer tool for locally displaying the 4 workcell camera feeds in real time (Figure 7.2), a simulated workcell that mimics the real workcell for safely testing motions prior to deployment on the real system, and utility functions such as a pixel-to-world transform using the depth camera and conversions between different pose representations.

PyReach also includes PyReach Gym, an application programming interface (API) modeled after OpenAI Gym [25]. Remote agents receive observations of the environment and request actions through this interface. In particular, at each time step with frequency up to 10 Hz, a remote agent can receive the joint angles and Cartesian pose of the arm, the binary state of the gripper (closed or open), and camera observations. The agent specifies an action to execute as a desired pose of the arm in either joint or Cartesian space and a desired binary state of the gripper.

Garment Folding Case Study: Problem Definition

We assume that the target folded configuration is known beforehand, that training and evaluation are performed in real (not simulation), that the hardware setup is as specified in Section 7.3, and that the garment stays the same during training and evaluation. The task is to iteratively execute two procedures in a loop: (1) crumple the T-shirt and (2) fold the T-shirt. Crumpling is performed via a series of 6 random drops of the T-shirt resulting in an average of 37.5% coverage (Section 7.5), where coverage is the fraction of the maximum 2D area the T-shirt is able to attain. The folding task is to manipulate the T-shirt toward the target configuration in Figure 7.3. We decompose the folding task into two subtasks: (1) *flattening*, i.e., spreading out from an initially crumpled configuration until the garment is smooth, followed by (2) *folding*, i.e., folding the t-shirt from initially flattened until sufficiently close to the target configuration. We measure folding accuracy with a combination of Intersection over Union (IOU) and wrinkle detection (Section 7.5).

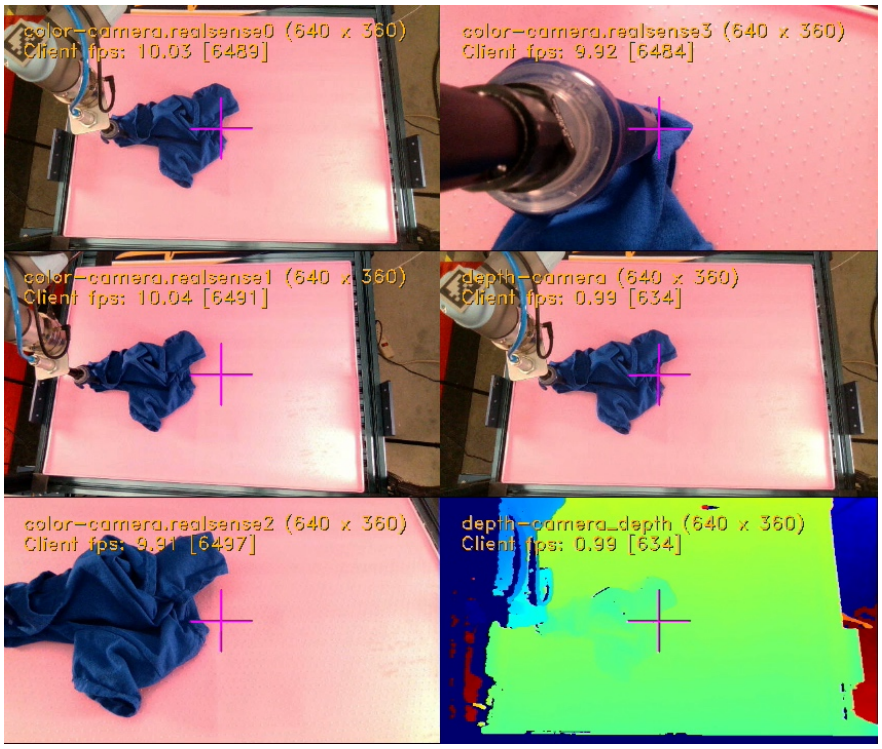


Figure 7.2: The client PyReach viewer, which updates the RGB images from the workcell cameras at 10 Hz and depth images at 1 Hz. Our algorithms use the overhead RGB images (top left panel). Note that the lower two panels on the right are from the same camera as the top left panel.

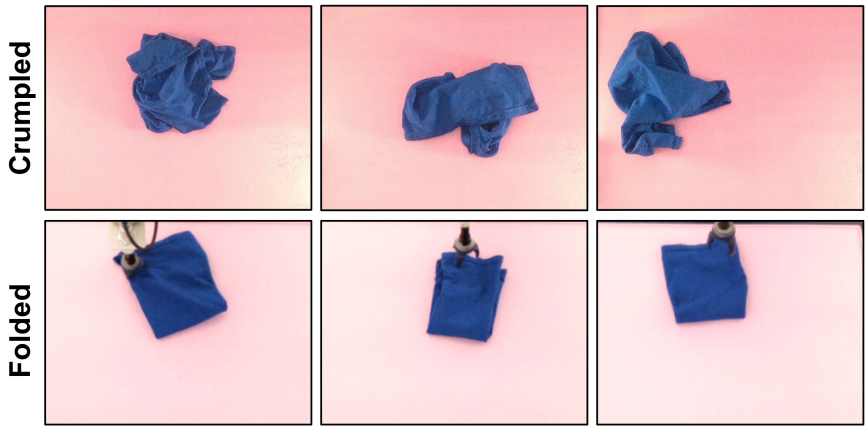


Figure 7.3: Examples of crumpled states (Row 1) and folded states (Row 2).

$\mathbf{o}_t$	Cropped RGB observation of the workcell state from the overhead Realsense camera at timestep t (Figure 7.2).
$\mathbf{a}_t$	The action at time t, expressed as a pick-and-place action (p_0, p_1) in pixel coordinates except in Section 7.4.
$\mathbf{o}_t^m$	Color-thresholded mask of the T-shirt analytically computed from $\mathbf{o}_t$.
$\text{com}(\mathbf{o}_t^m)$	A function that returns the *visual* center of the T-shirt.
$\text{c}(\mathbf{o}_t^m)$	A function that computes the 2D fabric coverage.
$\mathcal{T}$	A template image of a fully flattened shirt in the workspace.

Table 7.1: Notation for Section 7.4.

7.4 Garment Folding Algorithms

Due to the unique challenges of the flattening and folding subtasks, we benchmark each subtask with its own set of algorithms. Hyperparameter and implementation details for all algorithms are available in the appendix, and notation for this section is defined in Table 7.1. With the exception of Drop, all actions are quasistatic pick-and-place actions from pick point p_0 to place point p_1, where p_0 and p_1 are specified as (x, y) coordinates in pixel space; see Appendix E.4 for implementation details.

Flattening: 4 New Algorithms

Learned Pick-Analytic Place (LP_0AP_1)

Inspired by prior work in imitation learning for fabric manipulation [227, 98], we develop an algorithm to learn pick points from human demonstrations. Since we empirically observe that human-selected pick points combined with analytic placing performs well on flattening, we propose only learning the pick points p_0 and analytically computing place points with the strategy in Section 7.4 to improve sample efficiency. While other work has considered learning a pick-conditioned place policy for fabric manipulation [268], we define analytic placing actions that make the pick-conditioned policy unnecessary. To handle the inherent multimodality in the distribution of human-specified pick points, we train a fully convolutional network (FCN) [233] to output a heatmap corresponding to probability density instead of

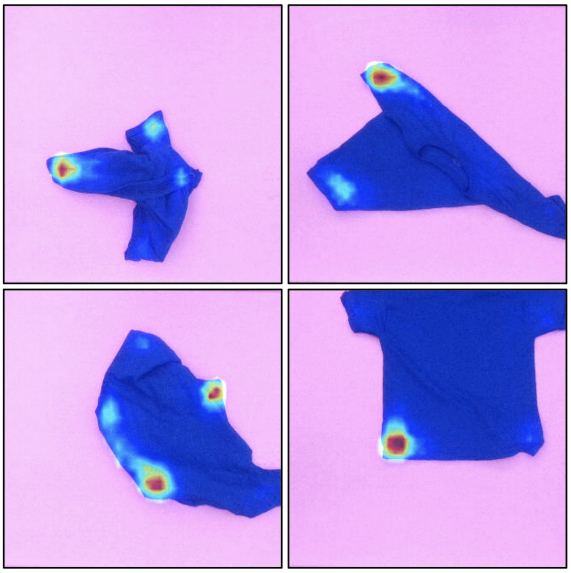


Figure 7.4: LP_0AP_1 pick point predictions on the test set. Bright red and yellow regions correspond to high probability pick points. The output heatmap is able to capture the multimodality in human actions.

regressing to an individual action (Figure 7.4). The FCN can be interpreted as an implicit energy-based model [75, 275] where the state and action pairs are the receptive fields of the network. As in DAgger [215], we reduce distribution shift by iteratively adding on-policy action labels to the dataset.

Learning Keypoints (KP)

This approach separates perception from planning and proposes to only learn the perception component. Specifically, we collect a hand-labeled dataset of images with up to 5 visible keypoints on the fabric corresponding to the collar, 2 sleeves, and 2 base corners (Figure 7.5). While the dataset generation policy is open-ended for this approach, we choose to first train an initial KP policy on random data (Section 7.4) and then augment the dataset with states encountered under the policy to mitigate distribution mismatch similar to DAgger [215]. We train a FCN with 3 output heatmaps to predict each of the 3 classes of keypoints separately. Using keypoint predictions, we propose an analytic corner-pulling policy inspired by [227] that iteratively moves the keypoints from their current positions to their respective locations on a template flattened shirt $\mathcal{T}$. To reduce ambiguity, we compute the rotation and translation of $\mathcal{T}$ that best matches the current state and first move the keypoint farthest from its target location to its destination. To our knowledge, the combination of the FCN for multi-class keypoint prediction, T-shirt template matching, and corner pulling is a novel flattening policy.

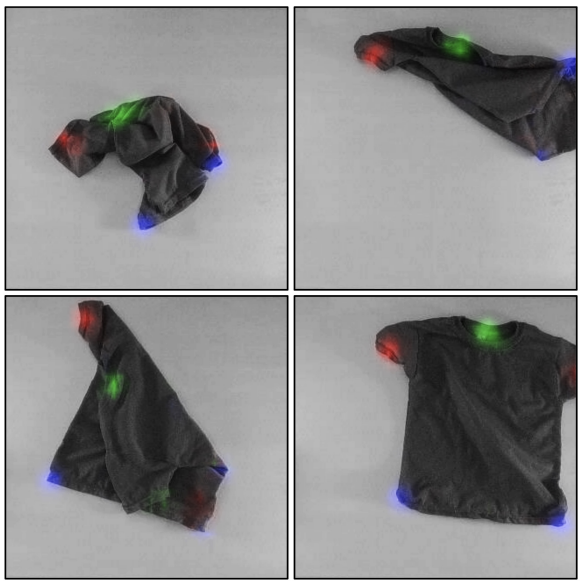


Figure 7.5: KP predictions on the test set. The predicted collar is colored green, the two sleeves are red, and the two base points are blue. Shirt images are shown in grayscale for viewing convenience.

Coverage Reward Learning (CRL)

This approach seeks to learn a reward function corresponding to fabric coverage $c(\cdot)$ from data and execute a policy using this reward function. We learn this reward with self-supervised learning and execute a greedy policy that seeks to maximize the 1-step reward at each time step. Specifically, we fit a Convolutional Neural Network (CNN) $R_\theta(\mathbf{o}_t, \mathbf{a}_t)$ to the scalar change in coverage (i.e., $c(\mathbf{o}^m_{t+1}) - c(\mathbf{o}^m_t)$) that results from executing action $\mathbf{a}_t$ on $\mathbf{o}_t$. At execution time we randomly sample thousands of pick points on the fabric mask $\mathbf{o}^m_t$ and place points in the workspace and select the action with the highest predicted change in coverage. To our knowledge, greedy planning over a learned model of coverage dynamics for fabric flattening is novel. Once again, dataset generation is a design choice; here, we opt for a random action policy (Section 7.4) to enable large-scale self-supervised data collection and increase data diversity.

Drop (DROP)

Inspired by Ha et al. [88], we investigate whether dynamic motions can leverage aerodynamic effects to accelerate the flattening of the shirt when combined with Approach 7.4. We propose a simple vertical drop primitive that grabs the visual center of mass $\text{com}(\mathbf{o}^m_t)$, lifts the shirt into the air, and releases. We profile the coverage dynamics of the drop and the LP_0AP_1 pick-and-place and run Q-value iteration to determine which primitive to execute (i.e., drop or pick-and-place) given a discretized version of the current coverage $c(\mathbf{o}^m_t)$. Q-value iteration

on the following reward function produces a policy that minimizes the total number of actions required to flatten the shirt:

$$r(s = \mathrm{c}(\cdot), a) = \begin{cases} -1 & \mathrm{c}(\cdot) < C \\ 0 & \mathrm{c}(\cdot) \geq C \end{cases}$$

where C is a coverage threshold defined in Section 7.5 and $\mathrm{c}(\cdot)$ is the discretized current coverage.

Flattening: 4 Baselines

Random (RAND)

As a simple baseline, we implement a random pick-and-place policy that selects p_0 uniformly at random from $\mathbf{o}_t^m$ and p_1 uniformly at random in the workspace within a maximum distance from p_0.

Human Teleoperation (HUMAN)

As an upper bound on performance and action efficiency, a human selects pick and place points through a point-and-click interface (see the appendix for details).

Analytic Edge-Pull (AEP)

We implement a fully analytic policy to explore to what extent learning is required for the T-shirt flattening task. The policy seeks to flatten the shirt by picking the edges and corners and pulling outwards. Formally, we sample p_0 uniformly from the set of points in the shirt mask $\mathbf{o}_t^m$ that are within a distance k from the perimeter of $\mathbf{o}_t^m$, where k is a hyperparameter. Given p_0, we compute p_1 by pulling a fixed distance l in the direction of the average of two unit vectors: (1) away from $\mathrm{com}(\mathbf{o}_t^m)$ and (2) toward the nearest pixel outside $\mathbf{o}_t^m$.

Learning an Inverse Dynamics Model (IDYN)

A inverse dynamics model $f(\mathbf{o}_t, \mathbf{o}_{t+1})$ produces the action $\mathbf{a}_t$ that causes the input transition from $\mathbf{o}_t$ to $\mathbf{o}_{t+1}$. Here we implement the algorithm proposed by Nair et al. [179], which learns to model visual inverse dynamics. Specifically, we approximate the dynamics with a Siamese CNN $f_\theta(\cdot, \cdot)$ trained on the random action dataset collected in Section 7.4. As in [179], the network factors the action by predicting the pick point p_0 before the pick-conditioned place point p_1 to improve sample efficiency. During policy evaluation, the inputs to the network are the current observation $\mathbf{o}_t$ and the template goal observation $\mathcal{T}$.

Folding Algorithms

Human Teleoperation (HUMAN)

As an upper bound on performance, a human chooses pick and place points for folding through a point-and-click interface.

Analytic Shape-Matching (ASM)

Since the folding subtask is significantly more well-defined than flattening, we investigate whether an open-loop policy computed via shape matching can successfully fold the shirt. We specify a fixed sequence of folding actions with a single human demonstration. During evaluation, we compute rotations and translations of the corresponding template images to find the best match with $\mathbf{o}_t$ and transform the folding actions in the demonstration accordingly.

Learned Pick-Learned Place ($\mathbf{LP_0LP_1}$)

This approach is identical to Section 7.4 but learns both pick points and place points, as the analytically computed place point is designed for flattening. Since folding demonstrations are difficult to obtain (the garment must be flattened first) and successful folding episodes are short-horizon and visually similar, we collect only two demonstrations and augment the data by a factor of 20 with affine transforms that encourage rotational and translational invariance.

Fully Autonomous Flattening with Analytic Shape-Matching (A-ASM)

The algorithms above are evaluated after the garment is fully flattened via human teleoperation to study the folding subtask in isolation. This approach, A-ASM, combines the best-performing autonomous flattening algorithm (i.e., $\mathrm{LP_0AP_1}$) with ASM (Section 7.4) to evaluate the performance of a fully autonomous pipeline for manipulating the garment from crumpled to folded.

7.5 Experiments

Experimental Setup

All actions executed on the robot are either a pick-and-place primitive (p_0, p_1) or a drop primitive (for the DROP algorithm). See Appendix E.4 or the code for the exact implementation details. During data collection, actions are chosen either autonomously (e.g., with RAND in Section 7.4) or by a human via a point-and-click graphical user interface (see the appendix). At execution time, actions are parameterized by outputs from trained models.

To improve the performance of the deployed flattening algorithms, we include two additional primitives: (1) a recentering primitive for when the shirt has drifted too far from the center of the workspace, and (2) a recovery primitive that executes a random action when the coverage is stalled for an extended period of time. See the appendix for ablation studies suggesting the usefulness of such primitives.

Flattening Metrics

We perform 10 trials of all flattening algorithms from an initially crumpled state (Figure 7.3). Crumpling is performed autonomously via a series of 6 actions, each of which grabs the T-shirt at a random point, quickly lifts it into the air, and releases, resulting in an initial coverage of 37.5% $\pm$ 14.9% over 45 trials. In Table 7.2 we report maximum coverage as a percentage of the pixel coverage of a fully flattened shirt, i.e. 47,000 pixels in the shirt mask $\mathbf{o}_t^m$. We also report the number of samples used to train the algorithm, the execution time per action, and the number of actions executed, where we allow a maximum of 100 actions but terminate early if a coverage threshold is reached (C = 45,000 pixels or 96% of maximally flattened).

Folding Metrics

We perform 5 trials of all folding algorithms from an initially flattened state. A-ASM initial states are flattened by LP_0AP_1 while all other initial states are flattened via human teleoperation. In Table 7.3 we report the number of actions and execution time per action, and we measure the quality of the final state against a goal configuration (Figure 7.3) according to two metrics: (1) intersection over union (IoU) and (2) a penalty for edges and wrinkles. IoU is calculated between the shirt mask and the goal template, after rotating and translating the goal to best match the shirt mask. The wrinkle penalty calculates the fraction of pixels in the interior of the shirt mask detected as edges by the Canny edge detector [34]. A high-quality folding episode achieves a high IoU score and low edge penalty; for reference, the scores for a fully folded goal image are provided in Table 7.3 as GOAL.

Flattening Results

See Table 7.2 and Figure 7.6 for results. We find that fully analytical policies such as RAND and AEP are unable to attain high coverage while HUMAN is able to consistently flatten the garment in 11.9 actions on average, suggesting the efficacy of the pick-and-place action primitive and the value of intelligently selecting pick points. Interestingly, we find that despite training an inverse dynamics model on nearly 4,000 real samples, IDYN is unable to outperform RAND. We hypothesize that the fully flattened goal image $\mathcal{T}$ provided as input is too distant from the encountered states, resulting in a data sample outside the training data distribution.

Table 7.2: Flattening results. We report the metrics in Section 7.5, where averages and standard deviations are computed over 10 trials.

Algorithm	**% Coverage**	**Actions**	**Dataset**	**Time/Act (s)**
RAND	55.0 ± 6.0	100.0 ± 0.0	N/A	23.9 ± 2.5
HUMAN	**97.7 ± 3.9**	11.9 ± 5.3	N/A	45.1 ± 18.6
AEP	55.3 ± 5.5	100.0 ± 0.0	N/A	24.6 ± 2.0
IDYN	57.0 ± 5.9	100.0 ± 0.0	3936	23.7 ± 3.7
KP	72.4 ± 9.2	100.0 ± 0.0	681	25.7 ± 2.7
CRL	73.8 ± 8.4	100.0 ± 0.0	3936	32.1 ± 5.3
DROP	**97.7 ± 1.3**	38.6 ± 20.6	524	25.7 ± 0.8
LP_0AP_1	**97.7 ± 1.4**	31.9 ± 17.2	524	25.6 ± 0.9

CRL is better able to leverage the large self-supervised dataset as it attains higher coverage, though it does require more time per action due to thousands of forward passes through the network during planning. However, since the dataset is generated by RAND, which achieves an average maximum coverage of only 55.0%, CRL has trouble producing high-quality actions in the high coverage regime, where it has encountered relatively little data. Modifications to the dataset such as actively interleaving data collection and training with policy execution is an interesting direction for future work. KP also achieves a higher maximum coverage than AEP and RAND, but it is prone to executing regressive actions that prevent it from maintaining this coverage. Results suggest that KP can be improved by (1) autonomous labeling, e.g. with fiducial markers, to avoid human error on challenging states with high self-occlusion, and (2) improvements to the analytic corner-pulling policy, which, for example, can struggle when all visible keypoints are positioned correctly but other keypoints are not visible.

We find that LP_0AP_1 significantly outperforms all other algorithms, rivaling HUMAN-level performance by consistently reaching the threshold coverage C in less than 3 times the amount of actions as HUMAN. We hypothesize that this is due to increased sample efficiency from analytic placing in conjunction with the modeling power of the FCN, which exhibits equivariance by sharing parameters for pixel predictions and is an implicit energy-based model like other state-of-the-art architectures [75, 275].

Finally, we find that DROP, which converges through Q-iteration to a policy that executes a drop if coverage is below 45% and LP_0AP_1 otherwise, is unable to improve upon LP_0AP_1. This may occur due to our modeling of the coverage dynamics of LP_0AP_1 as the same regardless of the current coverage, whereas in reality, LP_0AP_1 improves coverage faster in lower-coverage states (Figure 7.6). Nevertheless, the DROP framework may be an effective way to combine multiple action primitives given more powerful dynamic primitives, such as bimanual actions that can better leverage aerodynamic effects [88].

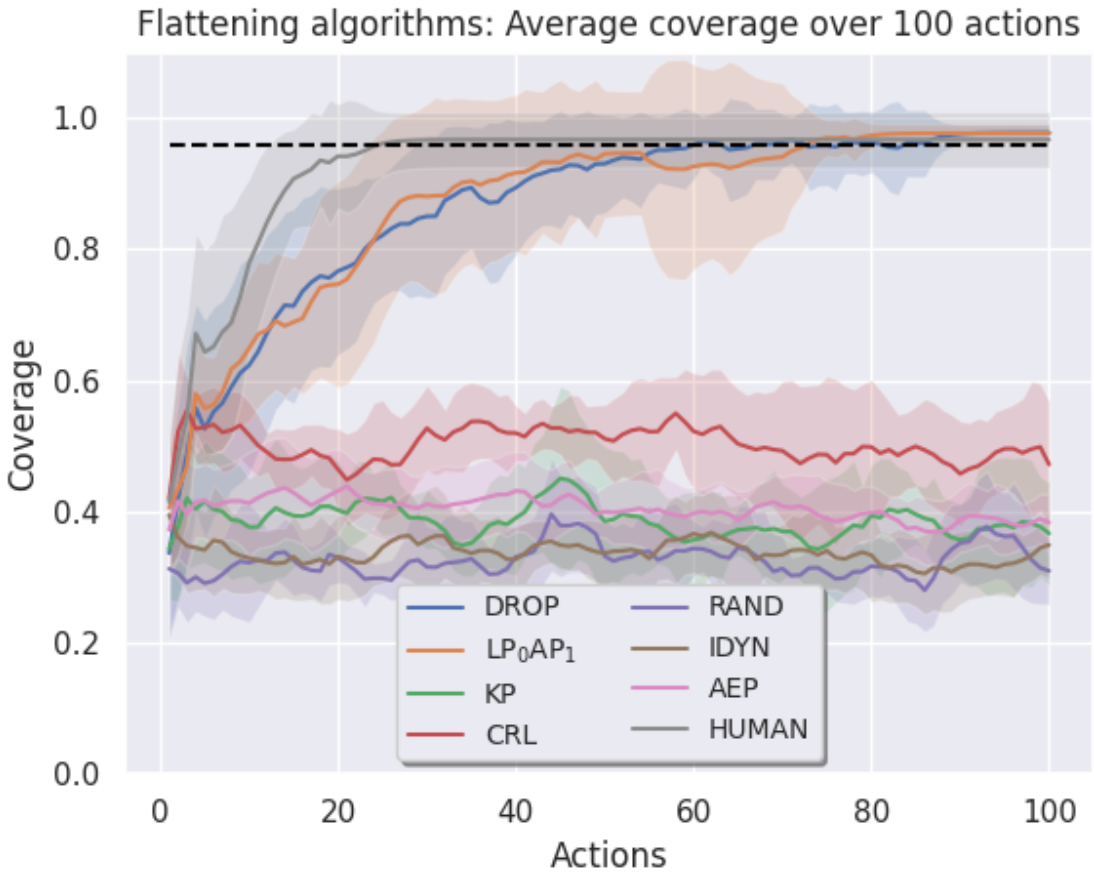


Figure 7.6: Coverage vs. time plot for the various flattening policies that we benchmark on the workcell, averaged across 10 rollouts. Shading represents one standard deviation, and the horizontal dashed line is the flattening success threshold (96%).

Folding Results

See Table 7.3 for results and Figure 7.7 for folding episodes. The folding subtask presents unique challenges: (1) data collection and evaluation require an initially flattened state, which is difficult to attain through a remote interface, (2) slightly incorrect actions can dramatically alter the fabric state, often requiring re-flattening the garment, and (3) the single-arm pick-and-place primitive is not well-suited for the precise manipulation required for crisp garment folding. Indeed, we find that even with folding optimizations to pick-and-place (Section 7.5), a human teleoperator attains only 76% of the goal IoU on average (Table 7.3). However, we find that both ASM and LP_0LP_1 are able to effectively leverage the primitive to achieve near human-level performance, where ASM performs similarly to LP_0LP_1. We also find that the fully autonomous pipeline A-ASM is able to reach similar per-

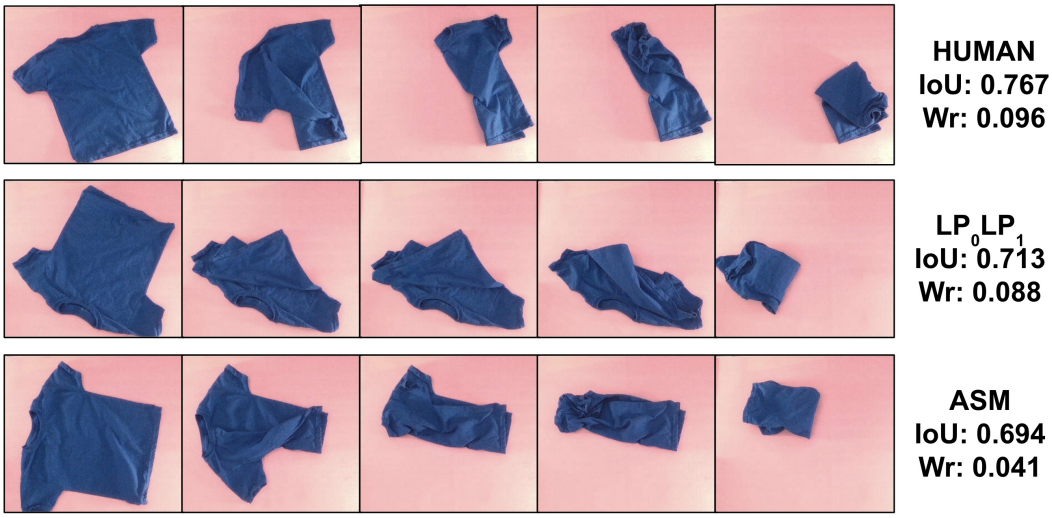


Figure 7.7: Representative episodes of the folding subtask executed by HUMAN (Row 1), LP_0LP_1 (Row 2), and ASM (Row 3). LP_0LP_1 and ASM achieve performance competitive with human teleoperation.

Table 7.3: Folding results. We report the metrics in Section 7.5, where averages and standard deviations are computed over 5 trials.

Algo.	**IoU (↑)**	**Wrinkle (↓)**	**Actions**	**Time (s)**
GOAL	0.98	0.093	N/A	N/A
HUMAN	**0.74 ± 0.06**	0.088 ± 0.023	4.4 ± 0.5	63.8 ± 15.
ASM	**0.69 ± 0.08**	**0.087 ± 0.038**	4.0 ± 0.0	35.1 ± 1.9
LP_0LP_1	0.68 ± 0.08	0.112 ± 0.032	4.0 ± 0.0	35.7 ± 1.3
A-ASM	0.62 ± 0.12	0.112 ± 0.038	4.0 ± 0.0	35.5 ± 1.7

formance from an initially *crumpled* state, setting a baseline score for the end-to-end folding task. Although ASM is open-loop and LP_0LP_1 learns from only 2 demonstrations, HUMAN cannot significantly outperform them due to the difficulty of correcting inaccurate actions in folding with only top-down pick-and-place actions. Further progress on the folding subtask will likely require both improved manipulation primitives and algorithmic innovations.

7.6 Conclusion and Future Work

In this work, we benchmark novel and existing algorithms for T-shirt smoothing and folding tasks on a remote hardware testbed. We find that policies that combine learning with

analytical methods achieve the highest performance in practice, suggesting the value of future work in this area.

The ability to access robot hardware remotely, an intuitive API, maintenance by dedicated staff, and the consistency of the task environment all contribute to quick and effective experimentation. On the other hand, onsite technicians have limited availability, variable-latency 2D camera projections are at times insufficient for fully understanding the scene, and manual resets (e.g., flattening the T-shirt) become difficult to perform, suggesting the importance of learning self-supervised reset policies [87].

Opportunities for future work include (1) further optimizing performance on the unimanual folding task, (2) evaluating alternative approaches such as continuous control, reinforcement learning, and different action primitives, and (3) evaluating each algorithm's ability to generalize to other garments with variation in color, shape, size, and material.

Chapter 8

FogROS2-SGC: Cloud Robotics with Secure Global Connectivity

In this chapter, we present FogROS2-SGC, a cloud robotics platform for securely connecting ROS2 networks in different physical locations. We study how this system may facilitate robot fleet learning with robots, compute nodes, and human supervisors distributed around the world.

8.1 Introduction

As robots are increasingly deployed worldwide, they require mechanisms to efficiently, reliably, and securely communicate with other robots, sensors, computers, and the cloud. The applications are broad, from mobile robots with changing IP addresses due to traveling through different networks, to a fleet of globally distributed robots learning collaboratively. Cloud and fog robotics [123] empower robots and automation systems to harness off-board resources in cloud-based computers. In prior work, we introduced FogROS2 [104], now an official part of the ROS2 ecosystem [76], to enable robots to execute modern compute and memory-intensive algorithms using on-demand hardware resources on the edge and cloud. However, ROS2 and FogROS2 assume all robots are locally connected and each robot has full access and control of other robots. Robots connecting to the cloud, a nearby computer on a different network, or a robot halfway around the world introduce additional challenges: (1) Robots that are accessible to other systems on the public internet may be vulnerable to unauthorized connections and data breaches. (2) The heterogeneity of interconnected devices, communication protocols, and configurations causes incompatibilities that hinder integration and operation. (3) The changing network topology of mobile robots and Unmanned Aerial Vehicles (UAVs) challenges their ability to stay connected. To illustrate some of these challenges (Fig. 8.1), consider:

(A) Security and inspection drones: Drones navigate a construction site and stream data to a central station that updates a dynamically-changing SLAM map [242]. As drones fly

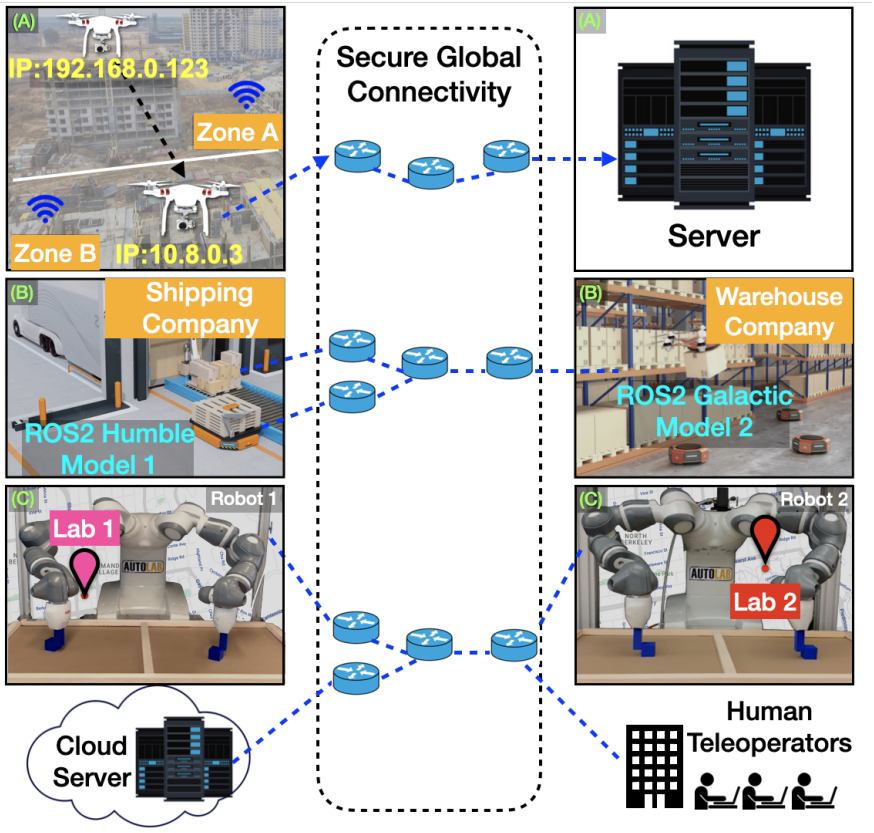


Figure 8.1: FogROS2-SGC enables Secure Global Connectivity for robots, allowing robots to communicate with other robots, computers, and the cloud through a standard ROS2 interface. With FogROS2-SGC, (A) drones navigating large construction sites can seamlessly communicate, even when their IP addresses are constantly changing due to switching Wi-Fi and cellular networks; (B) shipping and stocking robots from different corporations can securely share only the required topics necessary to facilitate the transfer of goods at a warehouse; and (C) globally distributed robots can participate in fleet learning. In experiments, we demonstrate FogROS2-SGC on Fleet-DAgger [99], a fleet learning algorithm, with 4 robot arms operating simultaneously in different locations.

through different cellular and Wi-Fi networks, their IP addresses change, but they should remain securely connected.

(B) Coordinating heterogeneous mobile robots in a warehouse: Robots belonging to different companies (e.g., shipping vs. warehouse) and of different makes and models hand off items between container and warehouse. Each robot has unique software packages and versions (e.g., operating systems and network protocols) and must communicate, but only a few selected topics are necessary for the handoff.

(C) Distributed fleet learning: Robot arms at different locations pool their data and collectively update a shared control policy. Robots unable to make progress can fall back on remote human teleoperators, using algorithms such as Fleet-DAgger [99].

To address these challenges, we present FogROS2-SGC (Secure Global Connectivity), an extension of FogROS2 that securely and reliably connects robots across different software components, network protocols, and physical locations. FogROS2-SGC enables disjoint ROS2 networks to connect to ROS2 topic interfaces named with *globally-unique* and *location-independent* identifiers. The robots using FogROS2-SGC can roam freely while staying connected because the identifiers are constant. They are 256-bit strings that are secure and anonymous to unauthorized attackers by construction —a brute-force attack would have to find a match among 10^{77} possibilities (a value close to the number of protons in the observable universe[1]). FogROS2-SGC adopts a *security-first* routing design, where only authenticated parties can connect to the robot and establish secure communication. In contrast to prior work such as SROS2 [167] and FogROS2 [104], FogROS2-SGC does not require merging distributed ROS2 networks, allowing robots to keep their ROS2 networks private and expose public topics only if explicitly configured. Providing fine-grain isolation and access control reduces the attack surface and enhances scalability.

FogROS2-SGC seamlessly integrates with ROS2 applications without code modifications via an SGC proxy. Its implementation and security policy configuration are agnostic to ROS2 distributions and their network transport middleware vendors. FogROS2-SGC is also compatible with non-ROS2 programs that interact with ROS2 components and can provide secure global connectivity to non-cloud servers and computers. Furthermore, since memory copy and synchronization operations are expensive for memory-constrained robots, the implementation of FogROS2-SGC processes can route data without performing unnecessary copies (also known as "zero copy").

Experiments suggest that FogROS2-SGC reduces the network latency of a cloud-based grasp planning application by 9.42× compared to unsecured rosduct [212]-rosbridge [59]. We also deploy FogROS2-SGC to simultaneously control a fleet of 4 robot arms in different physical locations with compute off-loaded to a server 3600 km away.

This paper makes the following contributions:

1. FogROS2-SGC, an extension of FogROS2 that connects disjoint ROS2 networks by assigning public ROS2 topics with globally-unique and location-independent identifiers.

[1]The Eddington Number [70] (N_{Edd}) is currently estimated to be 10^{80}.

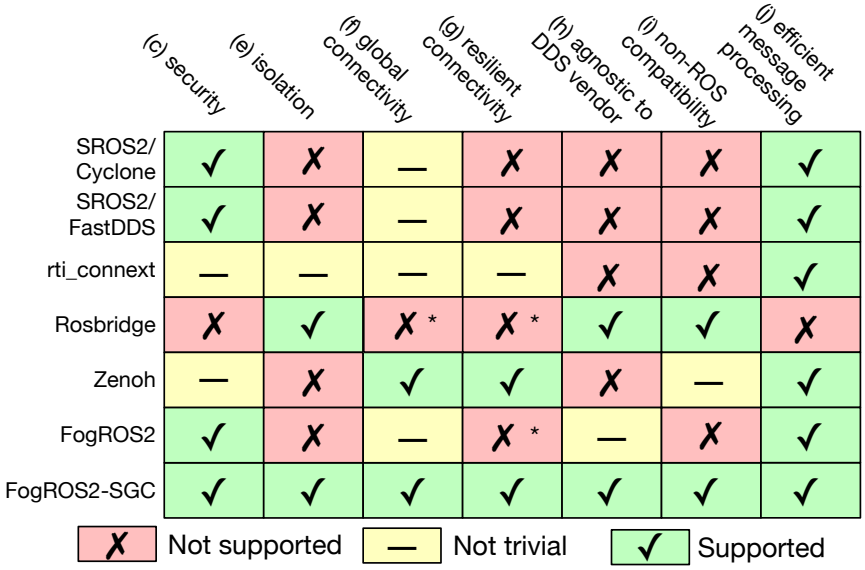


Figure 8.2: **Comparison of FogROS2-SGC with other distributed ROS2 systems.** In this table, we compare the feature support of different distributed ROS2 systems with the features in Section 8.3. Some features can be supported but require non-trivial effort beyond changing the configuration. For example, both the routing service in rti_connext and discovery server in FastDDS/SROS2 support global connectivity but require manually modifying routing rules or setting up a point-to-point VPN when a new node joins [223]. Rosbridge and FogROS2 support only unidirectional global and resilient connectivity (marked with *), meaning that one side of the communication must have a fixed IP. In contrast, the identifier-based routing of FogROS2-SGC allows either side to have a dynamic IP address.

2. Method for secure and efficient routing with FogROS2-SGC.
3. A Rust implementation of FogROS2-SGC that uses zero-copy message processing and asynchronous network operations for robots with memory and compute constraints.
4. Evaluation of FogROS2-SGC on cloud robotics applications (vSLAM, grasp planning, motion planning, simultaneous fleet control) demonstrating up to 9.42× latency reduction and enhanced usability.

8.2 Related Work

James Kaufner introduced the term 'Cloud Robotics' in 2010 [123]. Cloud and fog computing have been applied to robotic tasks such as grasp planning (Tian et al. [248], Kehoe et al. [122], and Li et al. [143]), parallelized Monte-Carlo grasp perturbation sampling (Kehoe et al.

[120, 121, 124]), and motion planning (Lam et al. [136]). Chen et al. [41] and Ichnowski et al. [104]. propose frameworks for offloading computation to resources on the edge or cloud, while Ichnowski et al. [105] and Anand et al. [8] present systems that leverage serverless computing [170]. Modern computing paradigms have enabled new applications such as multi-robot interactive fleet learning (Swamy et al. [245], Hoque et al. [99]) and remote sharing of robot systems (Tanwani et al. [246], Bauer et al. [17]).

Remote interactions between robots and the cloud raise security, compatibility, and connectivity challenges for robots. Virtual Private Networks (VPNs) are the most common approach for establishing secure communication between robots and the cloud for both ROS and ROS2 (e.g., Lim et al. [144]). Establishing a VPN link between a robot and the cloud is a complex process [90]. FogROS [41] and FogROS2 [104] automate the certificate generation and VPN setup. SROS2 [167] is an alternative approach to securing ROS2 communication that enforces access control of ROS2 topics. However, it requires DDS-dependent discovery mechanisms to ensure connectivity. Discovery mechanisms for DDS (such as the discovery server for FastDDS [71] and the RTI routing service for RTI Connext [218]) are vendor-specific and not compatible with other DDS implementations. Zenoh for ROS2 [107] is integrated with CycloneDDS to enhance peer-to-peer connectivity, but it is not compatible with other DDS implementations. ROS Remote [199] by Pereira et al. and MSA [272] by Xu et al. propose alternative protocols to unify cloud-robot communication. However, alternative protocols require modifications to ROS applications and are not compatible with ROS2. Finally, rosbridge [59] proposed by Crick et al. is widely adopted by both ROS1 and ROS2 to allow non-ROS software to interact with ROS2 nodes. It can also be used to bridge two non-compatible and remote ROS applications when used in conjunction with rosduct [212]. However, rosduct and rosbridge have significantly high message latency when the message size is large (e.g., images). A summary of how FogROS2-SGC differs from related work can be found in Fig. 8.2.

8.3 Ten FogROS2-SGC Features

FogROS2-SGC extends FogROS2 to address the Secure Global Connectivity (SGC) problem of securely and reliably connecting globally distributed robots, sensors, computers, and the cloud. We enumerate 10 new features to differentiate from related libraries and alternative approaches.

Globally identifiable addresses FogROS2-SGC enables a scalable number of ROS2 networks to publish a subset of ROS2 topics to other disjoint ROS2 networks around the globe. In scenario (C) from Fig. 8.1, the robot arms are located at different geographic locations with different local ROS2 networks. FogROS2-SGC allows remote human teleoperators to operate the robot arms as if the arms are connected to local networks. FogROS2-SGC also allows disjoint robots to publish to the same local ROS2 topics with globally unique and identifiable addresses.

Transparency to ROS2 applications ROS2 modularizes a robotics application into *nodes*, and connects the nodes into a graph. Nodes communicate with each other through a publish-subscribe (pub/sub) system, where publisher nodes send messages to *topics*, and nodes subscribed to these topics receive these messages. FogROS2-SGC adheres to the abstractions and interfaces of ROS2. ROS2 applications interact with remote nodes as if they are nodes on the same robot or subnetwork.

Communication security FogROS2-SGC guarantees that no unauthorized attacker can eavesdrop or tamper with ROS2 messages. Authorization is identified by user-configured cryptographic keys. In all three scenarios from Section 8.1, the robots communicate across wide-area networks with untrusted infrastructure. FogROS2-SGC prevents attackers from accessing any content in ROS2 messages and differentiates authentic robots from spoofing attackers.

Global anonymity Authorized participants can deterministically derive global identifiable addresses with ROS2 topic information and cryptographic secrets. Attackers cannot reverse any information used to recover addresses or topics. FogROS2-SGC prevent attackers that know part of the ROS2 topic information from deducing the global address. For example, the attackers who know the topic name and type information cannot guess the address, because they lack the author information and security credentials of the ROS2 node.

ROS2 network isolation and topic-level access control FogROS2-SGC connects robots without merging distributed ROS2 networks. Every robot can have an arbitrary number of private ROS2 topics and only public interfaces are shared with other authorized ROS2 networks. Other ROS2 nodes interact with these public interfaces just as they interact with a local ROS2 topic. This protects the privacy of the robot and prevents unintended messages from being shared with other disjoint networks of the system. For example, in scenario (B), a delivering robot from one company and receiving robot from another may have some proprietary topics that are kept private from each other. FogROS2-SGC isolates the topics private to each robot.

Global connectivity Some robots are connected to subnetworks that are not directly accessible from the outside. For example, robots in scenario (C) are in local area networks behind Network Address Translation (NAT). NAT allows multiple robots to share the same IP, but the translation is dynamic and ROS2 nodes outside cannot directly access the robots. FogROS2-SGC can connect ROS2 nodes that are behind firewalls and NAT.

Seamless and resilient connectivity to network dynamism FogROS2-SGC adapts to the dynamic network behaviors of drones and mobile robots. FogROS2-SGC does not rely on static IP addresses to identify the robots because such addresses are usually bound

to a physical location. Adding or reconnecting to robots should not restart the ROS2 node entirely, as this causes service interruptions and failures.

DDS-agnostic compatibility ROS2 adopts the Data Distribution Service (DDS) as its underlying network transport middleware to marshal, unmarshal, and exchange messages. ROS2 supports different DDS implementations, such as CycloneDDS [22], FastDDS [71], and RTI Connext [218]. However, a warehouse in scenario (B) may have robots running different versions of ROS2 and DDS. FogROS2-SGC is DDS-agnostic by leveraging ROS2 abstractions and not using any DDS-specific interfaces.

Compatibility with non-ROS2 software FogROS2-SGC allows non-ROS2 software to interact with ROS2 nodes. This principle is inspired by rosbridge [59]. Besides common transport protocols, while ROS2 officially supports C++ and Python, FogROS2-SGC allows programs to use gRPC [86], the most widely used Remote Procedure Call (RPC) framework that can run in heterogeneous environments and popular programming languages (including Go, Java, Javascript, PHP, and Rust) to control the robots.

Efficient message processing and routing ROS2 messages are buffered in memory to be processed by FogROS2-SGC. Because robots often have memory and compute resource constraints, FogROS2-SGC is memory-efficient by reducing unnecessary message copying and memory synchronization. Since FogROS2-SGC requires frequent exchange of packets to and from the network, network operations (such as `send` and `recv`) are not on the critical path of message processing.

8.4 FogROS2-SGC Design

See Fig. 8.3 for system architecture. FogROS2-SGC sends messages via a globally unique identifier. This identifier is unique to a robot and topic pair; thus, it can be used for sending and receiving messages regardless of robot location or network address. The identifier is secure, and communication is encrypted, meaning only authorized robots and nodes can access messages from its referenced topic. To implement routing based on the identifier, FogROS2-SGC consists of two main software components—(1) a router, responsible for securely routing messages between other routers and nodes, and (2) a proxy, that converts between ROS2 messages and the secure routers. As robots can be compute- and memory-constrained, FogROS2-SGC provides an efficient implementation.

Global Addressability

Maintaining a globally unique identifier enables the identification of a specific robotic component across subnetworks. FogROS2-SGC uses ROS2 topics as the minimal granularity for the global identifier because a topic is an interface to ROS2 nodes, and a ROS2 node can

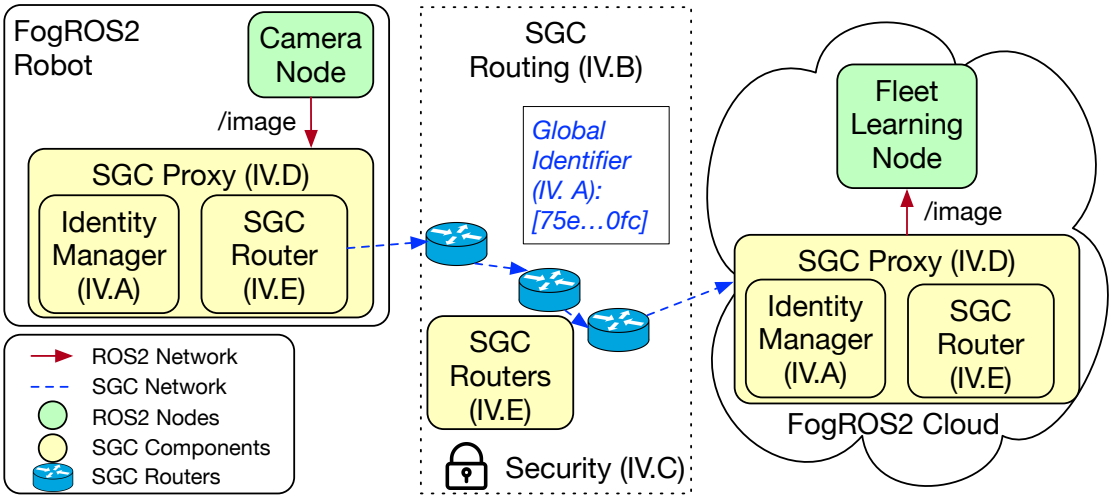


Figure 8.3: **System overview of FogROS2-SGC's architecture** showing a connection between a robot camera stream (on the ROS2 topic `/image`) and the cloud. The FogROS2-SGC assigns the ROS2 topic `/image` an anonymous, globally-unique and location-independent 256-bit identifier `[75e...0fc]` (truncated for brevity). The messages between identifiers are securely routed with SGC router.

publish or subscribe to multiple ROS2 topics at the same time. For example, a ROS2 vSLAM node in openVSLAM [242] has four ROS2 topics for camera information, video streaming, output localization, and mapping information. These ROS2 topics expose standardized interfaces with fixed message types. Users can limit the exposure of the ROS2 network by allowing only parts of the interface to be public. Partitioning public and private interfaces also enhances privacy and isolation, prevents unintended message exchanges, and reduces communication overhead.

The identifier is designed to be unique, deterministic, and location-independent. To avoid name collisions, every identifier has 256 binary bits, leading to 2^{256} possible identifiers. Instead of letting users decide, all identifiers are cryptographically derived from the metadata of the ROS2 topics by an identifier manager in SGC proxy. The SGC proxy collects metadata such as the ROS2 node's name, author, maintainer, interface, and description from standard ROS2 interface and user configuration file. The metadata also has a unique string in case the user needs to deploy the same topic at different locations. Every topic has an associated security certificate in X.509 [178] to verify the identity of those who want to publish or subscribe to the network. All the metadata is serialized and converted into a 256-bit string using SHA-256 [239], a widely used cryptographic hashing algorithm that maps arbitrary lengths of text to almost-unique 256-bit binary strings.

Security Analysis: The hashed string is suitable for use as the globally unique identifier for the followingreasons: (1) **Deterministic:** The hash is deterministic so that every party holding the same metadata can derive the same hash value and thus the same global identifier.

(2) **One-way:** SHA-256 is a one-way function, so the attacker cannot deduce or reverse the original metadata from the 256-bit identifier. (3) **Avalanche effect:** A small change to the original metadata leads to a new hash value that appears unrelated to the original hash value. (4) **Large namespace:** There are 2^{256} possible identifiers and it has been proved to be computationally intractable to find two messages with the same hash. Verification of these guarantees can be found in Appel [11].

Location-Independent Routing

Although having all identifiers in the same globally-flat namespace protects the privacy of the node's identity information and physical location, the identifiers do not carry any routing information. Flipping a bit in the identifier may lead to a drastic change in its physical location, or from existent to nonexistent. Therefore, securely routing messages between flat identifiers is a challenging problem. To solve this problem, FogROS2-SGC consolidates and extends the Global Data Plane (GDP) [173], a peer-to-peer network that routes messages between location-independent identifiers. The routers are set up by the user and peer-wise connected into a routing graph; robots do not need to know other robots' addresses as long as there is a connected routing path. The routers can be any machine that has network and general compute capabilities, such as an edge computer or a cloud server. Every router stores the mapping between the identifiers and the corresponding routing information of the identifiers in Routing Information Base (RIB).

A joining robot or router broadcasts an *advertisement* packet that announces the existence of the identifier and the routing information to the robot. The packet format is aligned with other FogROS2-SGC packets in Fig. 8.5. Other routers store the routing information in RIB and broadcast the advertisement packet. Routing is achieved by looking up the destination routing information in the RIB and forwarding to that destination.

Fig. 8.4 illustrates a step-by-step example of a publishing and subscribing `/camera` topic with FogROS2-SGC. The figure assumes that all the connections between routers are established. This can be achieved through configuration or dynamic node discovery [175]. The steps are:

1. The robot SGC proxy P1 generates an advertisement message for the ROS2 topic `/camera` and sends it to Router 1.

2. After verifying the advertisement message, Router 1 records the advertisement in its RIB and forwards the topic information to Router 2. There can be multiple routers between Router 1 and Router 2.

3. Cloud SGC proxy P2 requests to subscribe to `/camera`, and the subscribe request is sent to Router 2.

4. The subscribe request from Router 2 is routed to Router 1 by checking the source information at Router 2's RIB. After verifying the request, Router 1's RIB records P2 as the data sink.

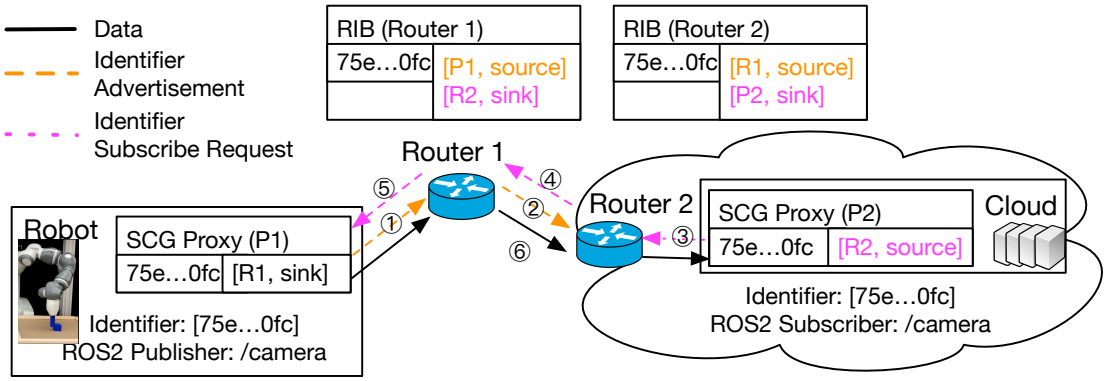


Figure 8.4: An illustration of how a routing connection is established between robot and cloud. The steps are further described in Section 8.4. (1,2) Advertisement generation and publish. (3,4,5) Subscribe request. (6) Data routing.

5. The subscribe request from Router 1 is routed to the robot by checking the source information at Router 1's RIB. If the destination is not found, the router broadcasts a query to other routers.

6. The robot's ROS2 publisher sends a ROS2 message to the proxy. The proxy forwards it to Router 1, Router 1 forwards to Router 2, and Router 2 to the cloud subscriber. At each hop, the messages are forwarded from source to sink.

Secure Communication

The security of the communication is achieved by using a secure network protocol between routers. We use Datagram Transport Layer Security (DTLS) [239] to provide communications privacy. The DTLS protocol provides secure and authenticated communication on User Datagram Protocol (UDP) and includes a built-in mechanism for dealing with lost or out-of-order packets. DTLS on UDP is well suited for latency-critical robotics communications systems, due to its lightweight nature and low overhead compared to transmission Control Protocl(TCP). The cryptographic algorithms used to secure ROS2 packet generation process can be found in Fig. 8.5. The message has the following security guarantees: **Confidentiality:** The ROS2 messages are encrypted with AES Encryption [61] to ensure that only parties with the correct cryptographic key can decrypt the original ROS2 message data. **Integrity:** The encrypted message is hashed by SHA-256 [11] so the receiver or third-party auditor can easily verify that the message is intact and no other attacker has tampered with the message. **Authenticity:** The hashed message is signed by the RSASSA-PSS [217] algorithm so that receivers can verify that the message is sent from an authorized sender.

To tailor the security with the communication patterns of robotics applications, FogROS2-SGC allows flexible peering with other routers or end points. One may choose to use a

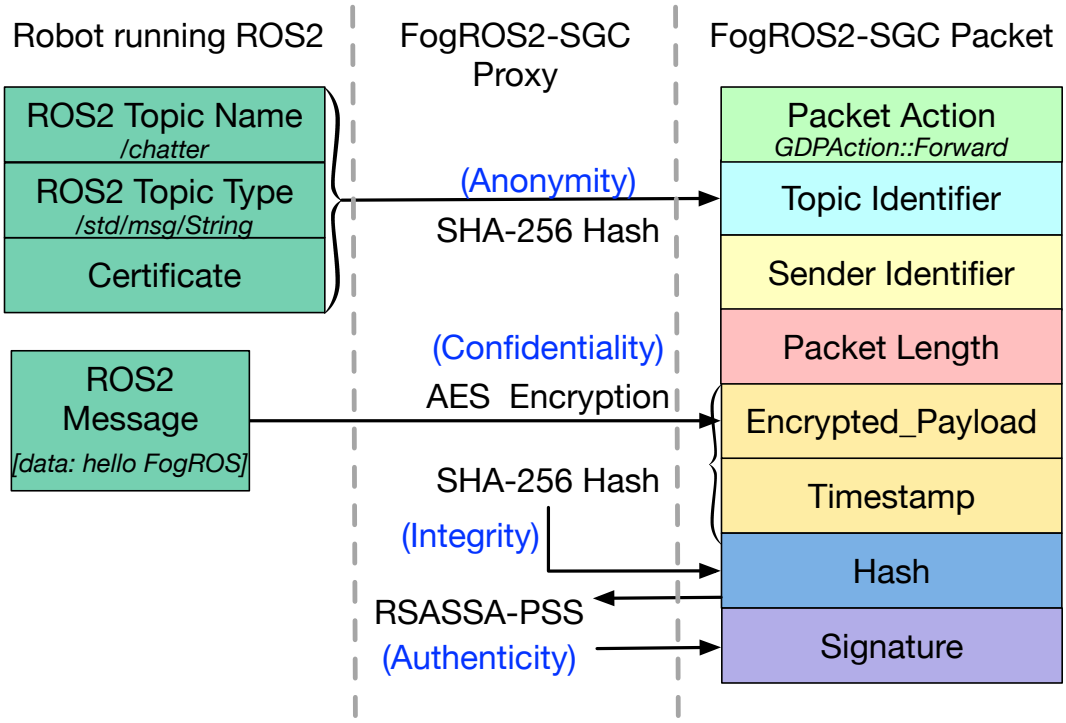


Figure 8.5: An illustration of the cryptographic tools used by SGC proxy to protect a ROS2 string message. The FogROS2-SGC Packet on the right is the message that is routed by FogROS2-SGC. The payload is encrypted to protect the confidentiality of the original ROS2 message. The encrypted data is hashed so that the receiver can verify the message is intact. The hash is signed with the sender's key so that the receiver can verify that the message comes from an authentic and authorized sender.

dedicated DTLS connection per ROS2 topic, which is ideal for large message payload and frequent communication (e.g., video streaming). One may also choose to use a shared DTLS tunnel, where multiple ROS2 topics share the same DTLS connection. Sharing the same connection reduces the cost of secure connection management and message processing, which is good for small message payloads and less frequent communication.

Transparent and Compatible SGC proxy

SGC proxy is the interface between FogROS2-SGC and the ROS2 network. In order to allow seamless integration with **any** unmodified ROS 2 application code and mainstream DDS vendors, SGC proxy converts between ROS2 communication and FogROS2-SGC communication bidirectionally. The user first identifies ROS2 topics that they wish to publish or subscribe through a configuration file. The proxy launches a local ROS2 publisher or subscriber for the corresponding topic. New messages from the local ROS2 network are ac-

tively subscribed to by the proxy, and sent to the FogROS2-SGC network. Once the verified subscribers receive the messages, they convert them to standard ROS messages and publish to their local ROS2 network.

To allow non-ROS2 programs to communicate with ROS2 nodes, SGC proxy converts ROS2 messages to a unified JSON-based message format in transit. As a result, FogROS2-SGC can be extended to a variety of protocols such as TCP, UDP, DTLS, TLS, and gRPC. Note, however, that some of the protocols need special handling to be aligned with FogROS2-SGC. For example, gRPC requires the IP addresses of both robot and cloud for bi-directional message passing.

Compute and Memory-Efficient SGC router

FogROS2-SGC can be deployed on low-power robots under memory and compute constaints, so an efficient implementation of routing algorithm in Section 8.4 is crucial to the overall performance of the system. Fig. 8.6 shows an architecture of SGC router. An idiomatic workflow of the router implementation is to (1) receive data from ROS2/network, (2) decide which network connection to forward, and (3) forward data to ROS2/network. Because FogROS2-SGC needs to be extensible to heterogeneous network protocols, the router needs to maintain many simultaneous network connections, ranging from ROS2's publish/subscribe protocol to general network protocol such as DTLS.

Because low-power robots run under memory constaints, memory copying operations and synchronization operations (such as mutex) are expensive. SGC router is implemented in Rust [166] to eliminate memory copying operations and the need for synchronization. Rust is a programming language that features a single ownership model: every data object has a single owner, and passing the data is moving the ownership from one variable to another. As a result, it prevents race conditions and reduces data copying by enforcing the passing of data objects by references instead of values.

Robots with few CPU cores usually have low network performance, because network operations are usually *blocking*, where the entire packet processing halts and waits for the network operations to finish. FogROS2-SGC improves CPU utilization by leveraging asynchronous Rust interfaces [252]. Asynchronous interfaces are non-blocking, removing the network operations out of the critical path of message processing.

8.5 Evaluation

We evaluate FogROS2-SGC on system benchmarks to show how it performs over alternative designs and on robotics benchmarks to show how robotics applications benefit from FogROS2-SGC. We use an Intel NUC with an Intel® Pentium® Silver J5005 CPU @ 1.50 GHz with a 5 Mbps network connection to act as the robot. The robot is connected with a Standard DS3 v2 cloud instance (4 vCPUs, 14 GiB memory) on Microsoft Azure. The robot is located at California (west coast of US), and the cloud server is located at Vir-

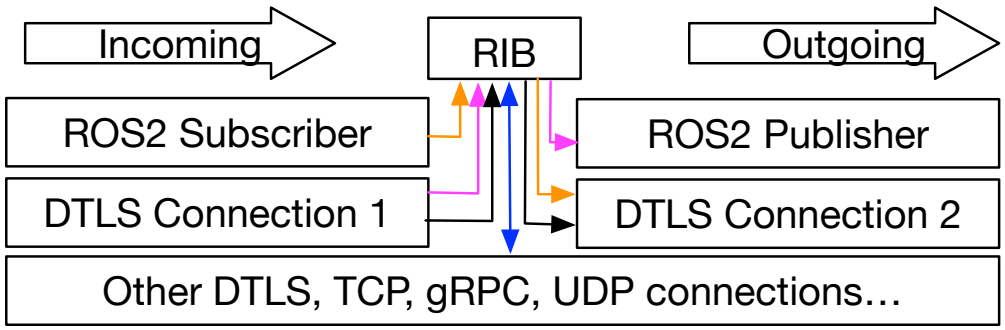


Figure 8.6: SGC router architecture. (**Orange**) Subscribe to a local ROS2 network and publish to FogROS2-SGC routing network. (**Magenta**) Receive from FogROS2-SGC routing network and publish to local ROS2 network. (**Black**) Intermediate SGC router that facilitates message routing. SGC router asynchronously reads, writes, and manages all the network connections. All the message passing (arrows) is zero-copy and does not require movement of actual messages.

ginia (east coast of US). We also demonstrate FogROS2-SGC on four physical robot arms running Fleet-DAgger [99], a multi-robot learning application.

System Benchmarks

We evaluate the performance of FogROS2-SGC's message processing latency and throughput against other distributed ROS2 systems. Messages are sent in binary with type `sensor_msgs/CompressedImage` and response with string type `std_msgs/String`. We compare against the following baselines (1) **VPN**: We use Wireguard VPN [265], which is the same VPN as FogROS2 [105] (2) **Rosbridge:** Rosbridge is the most commonly used websocket proxy that allows non-ROS code to interact with ROS code. We use Rosbridge in combination with Rosduct in the same way as in FogROS [41]. (3) **Capsule:** We use Capsule, a software switch inspired by Netbricks [192], to emulate the design of FogROS2-SGC. We also implement rosduct [212] in ROS2 that converts between ROS2 and network traffic. The detailed description and implementation can be found in FogROS-G [40]. **FogROS2-SGC** uses the default DTLS network protocol. We include **FogROS2-SGC-TCP** that uses TCP instead of DTLS as a variant.

ROS2 Message Latency: We measure the Round Trip Time (RTT) between when a robot publishes a ROS2 message to the cloud and when data is received by the cloud, which echoes a short message on a separate ROS2 topic. The RTT also includes the time of parsing the messages and analyzing the latency. The result can be found in Fig. 8.7. FogROS2-SGC with DTLS has similar performance as VPN, which has 0.076s round trip latency for small messages. FogROS2-SGC is 10.2% faster than VPN for 8000 byte messages (0.088 vs 0.097). FogROS2-SGC is 19× faster than rosduct-rosbridge (0.088 vs 1.67). There are

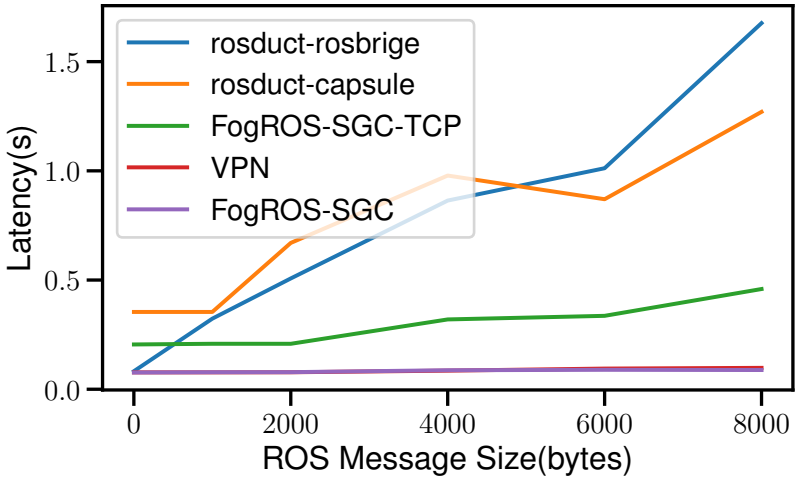


Figure 8.7: **Message round trip latency to the cloud** (lower is better). Latency is averaged over more than 50 packet window. FogROS2-SGC is 19 times faster than rosbridge baseline for 8000 byte message.

Protocol	Throughput (msg/second)
Original ROS	330.43
SROS2	320.17
Rosduct-Rosbridge	152.79
FogROS2-SGC-TCP	268.03
FogROS2-SGC	320.40

Table 8.1: **Message throughput evaluation of FogROS2-SGC** (higher is better). Every message is 1000 bytes. The throughput of FogROS2-SGC is near native performance while adding secure and global connectivity and 2.1 times higher than rosbridge.

two reasons for this: (1) Rosduct is implemented in Python and provably slower than Rust. It uses blocking network operations while FogROS2-SGC uses non-blocking asynchronous network operations for sending and receiving data. (2) Rosbridge requires seralization of binary messages in JSON, which require more bytes and lead to larger messages.

ROS2 Message Throughput: Message throughput is measured by the number of messages processed per second. Different from other experiments, throughput is measured on the local area network connected with Ethernet, in order to prevent network bandwidth from being the bottleneck. Table 8.1 shows the message processing throughput. FogROS2-SGC achieves near-native throughput as ROS2 and incurs only 3% overhead due to the security and conversion to a unified message format. FogROS2-SGC has 2.1× higher throughput than rosbridge, because rosbridge requires more bytes to serialize binary strings.

	vSLAM		Grasp Planning		Motion Planning	
Scenario	angle=45,lap=-(1em)fr1/xyz1	angle=45,lap=-(1em)fr1/loop	angle=45,lap=-(1em)raw matrix	angle=45,lap=-(1em)Compressed	angle=45,lap=-(1em)Apartment	angle=45,lap=-(1em)Cubicle
rosduct-rosbridge	10.31	10.29	20.3	13.67	0.08	0.08
VPN	1.16	1.45	**5.7**	1.47	0.07	0.07
FogROS2-SGC-TCP	1.19	1.57	8.4	1.58	0.07	0.07
FogROS2-SGC	**1.15**	**1.42**	-	**1.45**	0.07	0.07

Table 8.2: **Network latency of FogROS2-SGC on cloud robotics applications** (lower is better) FogROS2-SGC is better than rosduct-rosbridge and VPN on vSLAM and compressed grasp planning. We conducted motion planning on other scenarios (Home, Twisty-Cool) and the latency is the same.

Startup and Advertisement Time: In a RIB that has 10,000 routing records, the average time for publishing a name to the RIB takes 4ms and subscribing to a name from RIB takes 2ms. The average startup time from starting a program to receiving the first message takes 2.4 ms.

Cloud Robotics Application Benchmarks

We evaluate the network latency of FogROS2-SGC with 3 example cloud robotics applications: SLAM with ORB-SLAM2 [176], Grasp Planning with Dex-Net [155], and Motion Planning with Motion Planning Templates (MPT) [103]. The detailed description of these benchmarks can be found in [41].

As detailed in Table 8.2, although FogROS2-SGC can scale to multiple robots and provide fine grained access control for the robots, it demonstrates even better point-to-point performance than VPN in the vSLAM and grasp planning experiments. FogROS2-SGC is 9.42 times faster than rosbridge-rosduct on compressed grasp planning images. However, FogROS2-SGC cannot reliably transmit large and uncompressed grasp planning matrices. The raw matrix after serialization is larger than 13MB. We observe a significant amount of lost and out of order messages because the default transport protocol of FogROS2-SGC is DTLS over UDP and the communication channel does not recover from lost and out of order messages. Although transmitting such large message within single ROS2 message is rare, users can choose other supported transport protocols (such as TCP, gRPC) to meet the requirement of their applications.

Case Study: Fleet-DAgger

We apply FogROS2-SGC to the control of a fleet of 4 physical robot arms, an increasingly relevant setting in robotics and the third motivating example in Fig. 8.1. We use the physical experiment setup from Fleet-DAgger [99], where each robot simultaneously performs an image-based block-pushing task (see Fig. 8.1C). The task is to repeatedly push a cube to a goal region randomly generated in the image, where a new goal is sampled from the reachable workspace upon reaching the previous goal. The 4 workspaces have an identical

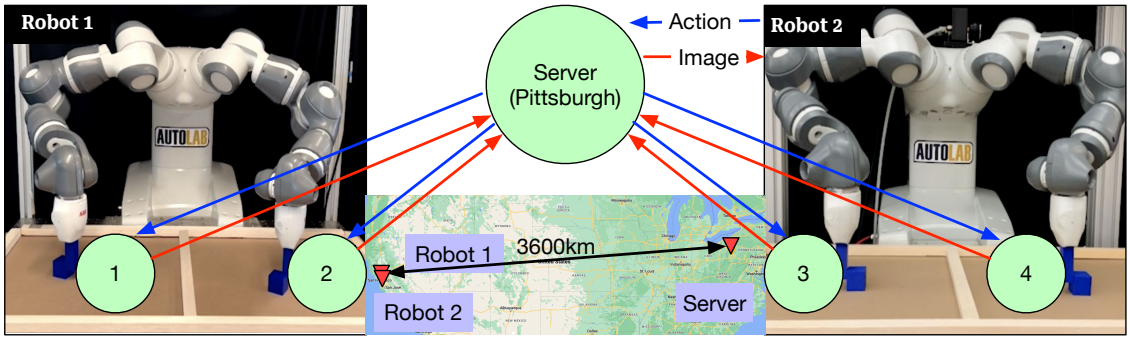


Figure 8.8: **The experiment setup of Fleet-DAgger**. Two ABB YuMi robots located in two separate buildings in Berkeley utilize computation from a server located in Pittsburgh for an image based block pushing task.

setup (but different block positions and goals) to enable the aggregation of each robot's data into a shared dataset and training of a single shared policy on this dataset, as is typical in fleet learning [99]. When autonomous control is unreliable, the robots fall back on and learn from remote human teleoperation, where global connectivity can dramatically increase the number of available humans. The arms belong to two bimanual ABB YuMi robots in two different labs about 1 km apart with separate local area networks. To test global connectivity, compute is off-loaded to a separate node in a third local area network at Carnegie Mellon University 3600 km away, where the robot nodes send images of the current state and receive actions to execute.

In a previous implementation, Hoque et al. [99] use Secure Shell (SSH) and Secure File Transfer Protocol (SFTP) to communicate between robots and the centralized compute node and Python multiprocessing to enable simultaneous execution. This approach requires storing all SSH credentials at a single node (a security concern), writing image data to the file system of all nodes at every timestep, complex asynchronous programming, and restricting all node locations to within the university campus firewall. To mitigate these issues, we (1) re-implement the communication system with ROS2 and (2) seamlessly connect all nodes with FogROS2-SGC with TCP by modifying only a single configuration text file on each node. Relative to the previous implementation, the FogROS2-SGC implementation reduces communication time by 64% (Table 8.3), where communication time includes image transmission latency and synchronization across all arms but not machine learning or arm motion. FogROS2-SGC also reduces communication time by 33% relative to the initial implementation even when the robots are in Berkeley, CA and the server is moved to Pittsburgh, PA. Note that the SSH method does not work between Berkeley and Pittsburgh due to university network firewalls [257]. A diagram of the system architecture is in Fig. 8.8.

Communication System	Server Location	Communication Time (s)
2*SSH + SFTP	Berkeley, CA	0.86
	Pittsburgh, PA	-
2*FogROS2-SGC	Berkeley, CA	0.31
	Pittsburgh, PA	0.58

Table 8.3: **Communication time of SSH+SFTP and FogROS2-SGC** (lower is better). FogROS2-SGC with TCP reduces the communication time per experiment step (i.e., one simultaneous action on the 4 arms) by 64% when compared to SSH+SFTP, and has 33% lower communication time than SSH+SFTP in Berkeley even if the server is moved to Pittsburgh. SSH does not work if the server is in Pittsburgh due to a university firewall restriction.

8.6 Conclusions and Limitations

We present FogROS2-SGC, an extension of FogROS2 that securely connects robotics components across different physical locations and networks. One limitation of FogROS2-SGC is that users are unable to use retransmission and Quality-of-Service (QoS) mechanisms provided by DDS for inter-ROS2 network communication. However, users can flexibly choose any supported transport protocol (e.g., TCP and gRPC). FogROS2-SGC also requires the intermediate routers to open certain ports for robots and services to connect.

Part IV

Conclusion

Chapter 9

Conclusion

9.1 Summary

In this dissertation, we present a comprehensive approach for scaling real-world robot learning to large-scale data collection and policy evaluation with robot fleets. As learned robot policies are still notoriously brittle, a popular strategy to bridge the gap and deliver the required level of reliability is to fall back to human control when needed. This approach, termed *supervised autonomy*, has received little attention in academia despite its prevalence in modern industrial settings. Supervised autonomy allows imperfect robots to deliver value today while simultaneously enabling them to improve over time and reduce their reliance on human supervision.

In Part I, we explore how supervised autonomy may be implemented with interactive imitation learning. In particular, robot-gated interactive imitation learning facilitates scalability by enabling robots to request supervision "on-demand," similar to a push notification on a smartphone. We propose LazyDAgger in Chapter 2 and ThriftyDAgger in Chapter 3, novel robot-gated interactive IL algorithms that strike a balance between the competing objectives of robot task performance and burden on the human supervisor. We also propose IntervenGen in Chapter 4, a system for further reducing burden on the human supervisor by synthetically generating large datasets of intervention behaviors from only a handful of human interventions.

In Part II, we study how on-demand supervision enables supervised autonomy with large robot fleets and multiple human supervisors, where robots can significantly outnumber humans. With fleet learning, each robot in the fleet can learn from the experience of all the other robots and continually improve over time. In Chapter 5, we propose the *interactive fleet learning (IFL)* formalism for the multi-robot, multi-human setting as well as Fleet-DAgger, a family of IFL algorithms that adapt algorithms like LazyDAgger and ThriftyDAgger to the fleet setting. We also propose an open-source software benchmark environment for systematically and efficiently evaluating the effectiveness of novel IFL algorithms (i.e., human-to-robot allocation strategies) with large-scale fleets of 100+ robots in simulation. In Chapter 6, we

develop a novel IFL algorithm for learning from heterogeneous human supervisors that may teleoperate in different ways. To do so, we propose a novel approach for quantifying uncertainty in energy-based models using Jeffreys Divergence [111] that is broadly applicable beyond the IFL setting.

Finally, in Part III, we propose and study systems for remote fleet supervision. Since modern networking technology enables real-time teleoperation across vast distances, the human supervisors in the IFL paradigm do not need to be physically present with the robot systems and may be located anywhere around the globe. Moreover, by connecting to cloud computing resources, robots no longer need to be limited to on-board memory and compute [123]. In Chapter 7, we perform a case study of fully remote robotics research with a prototype industrial robot workcell, using it to perform the first systematic benchmarking of fabric manipulation algorithms. In Chapter 8, we propose a cloud robotics platform for seamlessly and securely connecting disjoint networks around the globe, enabling real-time distributed robot fleet learning.

9.2 Limitations and Opportunities for Future Work

Like all research, despite its contributions, each part of this thesis has limitations. Addressing these limitations poses exciting opportunities for future work.

Scalable Interactive Imitation Learning

The main limitation of LazyDAgger (Chapter 2) and ThriftyDAgger (Chapter 3) is in their intervention criteria for actively soliciting human help. While they compare favorably to prior approaches, there is still room to improve. Uncertainty quantification and out-of-distribution detection in machine learning are still open challenges: models are generally ignorant of their own blind spots, as this is an "unknown unknown." For this reason, one goal of the IFL Benchmark proposed in Chapter 5 is to facilitate the development and standardized comparison of new intervention criteria. Continuing to iterate on these criteria is an active area of research [147, 210]. One interesting direction is to provide statistical performance guarantees and rigorous confidence intervals on the estimated uncertainty, e.g., with conformal prediction [9]. Another interesting direction is to optimize the intervention points themselves with interactive reinforcement learning [150].

Central to the motivation of LazyDAgger, ThriftyDAgger, and IntervenGen (Chapter 4) is the idea that all data is *not* created equal. It matters *where* in state space the data is collected; given a finite budget of human data, this motivates active data acquisition in the most informative and useful regions of state space with respect to the task and current robot policy. Often there are two broad classes of states within a task: "bottleneck" regions with narrow action tolerances (e.g., threading a needle), and more forgiving regions where coarse imitation suffices (e.g., moving the end effector in free space). There have been some

promising recent works in this direction that explicitly reason about this distinction between coarse and fine actions [114, 19, 235].

Interactive Fleet Learning

The IFL formalism (Chapter 5) has modeling assumptions that limit its generality: (1) all robots and humans operate in the same state and action space, (2) robots are independent and do not coordinate with each other, (3) humans have perfect situational awareness [46] and can move to different robots without any switching latency, and (4) timesteps are synchronous without network latency or other communication issues [131]. Lifting these assumptions are fruitful directions for future work. For instance, rather than requiring the human to provide low-level teleoperation in the same action space as that of the robots, natural language is a more flexible and intuitive modality for corrective feedback. While natural language feedback is difficult to ground into robot control, recent works have made progress in this direction [152, 60, 236].

As the IFL formalism does not specify a specific policy representation, it can be flexibly changed to other representations. For instance, Implicit IFL (Chapter 6) changes the IFL policy from an explicit one to an implicit one by extending Implicit Behavior Cloning [75]. A compelling newer alternative for representing multimodal data distributions is Diffusion Policy [43]. Another option is to learn a multi-task, cross-embodiment policy such as RT-X [189]: this further facilitates scalability by reusing data across robot embodiments and task specifications.

Systems for Remote Fleet Supervision

One design decision in the systems of Chapters 7 and 8 is a centralized model, in which the data from the different robots in the fleet are pooled into a single location for model training and supervisor allocation. An interesting alternative to consider is a decentralized or federated model [4, 131], especially for applications where privacy and security are paramount concerns.

Another critical component in the design of a remote fleet supervision system is the teleoperation interface. In imitation learning, the downstream robot control policies can only be as performant as the human teleoperation permits. Systems with high-dimensional input such as vision-based teleoperation [92] are an important direction for future work, both for enabling high-dimensional control and facilitating large-scale data collection. Other directions include getting haptic feedback [188] back to the human teleoperator, as well as designing intuitive interfaces for fleet supervision with insights from human-computer interaction research.

9.3 Broader Perspective on Robot Learning

At the time of writing this thesis in 2024, robot learning systems have begun to graduate from academic labs into real-world environments: autonomous taxis on the roads, humanoid and bin-picking robots on the factory floor, and more. Imitation learning has become the dominant paradigm for training these systems.

There is a palpable enthusiasm in the air and widespread hope that a watershed moment for robotics is just around the corner via large-scale supervised learning, just as it has transformed natural language processing and computer vision in the last couple years. Time will tell whether this hope will materialize, and if so, over what timescale. My personal stance is one of cautious optimism: there has certainly been dramatic progress in recent years, but much research remains to be done before we have truly general-purpose robot intelligence.

A few directions stand out to me as particularly promising lines of investigation. Imitation learning has indeed produced very impressive results in the last couple years. Policy representations like Diffusion Policy [43] that facilitate multimodal and high-dimensional output are capable of remarkably dexterous control with modest amounts of human data. Moreover, end-to-end imitation learning with large-scale data has begun to demonstrate impressive levels of reliability in autonomous driving. As such, investigating the data scaling laws for imitation learning in various domains (not limited to autonomous driving) will be crucial. Toward this goal, it will likely be helpful to have clever teleoperation interfaces that may facilitate large-scale robot data collection such as UMI [44].

Sim-to-real reinforcement learning has also been very impressive in recent years, particularly for biped and quadruped locomotion [134, 207], which until recently has relied on highly engineered model-based control. Can this recipe extend to manipulation, where accurate state estimation for relevant objects in the scene becomes crucial?

Finally, it will be very interesting to investigate the capabilities, properties, and scaling laws of vision-language-action models: large models that are first trained on Internet-scale text and image data and then fine-tuned on a much smaller dataset of robot trajectories [26]. If large language models are indeed general pattern machines [172], perhaps robot trajectories are just sentences in a new language.

It is an exceptionally exciting time to be in robotics, and I feel very fortunate to be working in this field. The recent breakthroughs in vision and language are still confined to the world of bits; robotics is a quest to unify the world of bits with our world of atoms. It is an incredibly difficult endeavor, but that makes it all the more appealing. "We can only see a short distance ahead, but we can see plenty there that needs to be done" [256].

Bibliography

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

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. "Do as I can, not as I say: Grounding language in robotic affordances". In: *Conference on Robot Learning (CoRL)*. 2022.

[3] Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan C. Julian, Sean Kirmani, Isabel Leal, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Sharath Maddineni, Kanishka Rao, Dorsa Sadigh, Pannag R. Sanketi, Pierre Sermanet, Quan Ho Vuong, Stefan Welker, Fei Xia, Ted Xiao, Peng Xu, Steve Xu, and Zhuo Xu. "AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents". In: *ArXiv preprint arXiv:2401.12963* (2024).

[4] Oguzhan Akcin, Pohan Li, Shubhankar Agarwal, and Sandeep Chinchali. "Data Games: A Game-Theoretic Approach to Swarm Robotic Data Collection". In: *Conference on Robot Learning (CoRL)* (2022).

[5] Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. "Transferring Dexterous Manipulation from GPU Simulation to a Remote Real-World TriFinger". In: *arXiv preprint arXiv:2108.09779* (2021).

[6] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. "Power to the people: The role of humans in interactive machine learning". In: *Ai Magazine* 35.4 (2014), pp. 105–120.

[7] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J. Grosz. "Interactive Teaching Strategies for Agent Training". In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.

[8] Raghav Anand, Jeffrey Ichnowski, Chenggang Wu, Joseph M Hellerstein, Joseph E Gonzalez, and Ken Goldberg. "Serverless Multi-Query Motion Planning for Fog Robotics". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021.

[9] Anastasios Nikolas Angelopoulos and Stephen Bates. "A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification". In: *arXiv preprint arXiv:2107.07511* (2021).

[10] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. "PaLM 2 technical report". In: *arXiv preprint arXiv:2305.10403* (2023).

[11] Andrew W Appel. "Verification of a cryptographic primitive: SHA-256". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 37.2 (2015), pp. 1–31.

[12] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[13] Saurabh Arora and Prashant Doshi. "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *arXiv preprint arXiv:1806.06877* (2018).

[14] Yahav Avigal, Lars Berscheid, Tamim Asfour, Torsten Kroger, and Ken Goldberg. "SpeedFolding: Learning Efficient Bimanual Folding of Garments". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022), pp. 1–8.

[15] J. Andrew Bagnell. *An Invitation to Imitation*. Tech. rep. CMU-RI-TR-15-08. Pittsburgh, PA: Carnegie Mellon University, Mar. 2015.

[16] Ashwin Balakrishna, Brijen Thananjeyan, Jonathan Lee, Felix Li, Arsh Zahed, Joseph E. Gonzalez, and Ken Goldberg. "On-Policy Robot Imitation Learning from a Converging Supervisor". In: *Conference on Robot Learning (CoRL)*. PMLR. 2019.

[17] Stefan Bauer, Felix Widmaier, Manuel Wüthrich, Niklas Funk, Julen Urain De Jesus, Jan Peters, Joe Watson, Claire Chen, Krishnan Srinivasan, Junwu Zhang, Jeffrey Zhang, Matthew R. Walter, Rishabh Madan, Charles B. Schaff, Takahiro Maeda, Takuma Yoneda, Denis Yarats, Arthur Allshire, Ethan K. Gordon, Tapomayukh Bhattacharjee, Siddhartha S. Srinivasa, Animesh Garg, Annika Buchholz, Sebastian Stark, Thomas Steinbrenner, Joel Akpo, Shruti Joshi, Vaibhav Agrawal, and Bernhard Schölkopf. "A Robot Cluster for Reproducible Research in Dexterous Manipulation". In: *arXiv preprint arXiv:2109.10957* (2021).

[18] Erik Båvenstrand and Jakob Berggren. "Performance Evaluation of Imitation Learning Algorithms with Human Experts". In: *Technical Report KTH Royal Institute of Technology Sweden*. 2019.

[19] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. "HYDRA: Hybrid Robot Actions for Imitation Learning". In: *Conference on Robot Learning (CoRL)*. 2023.

[20] Christopher M. Bishop. "Mixture Density Networks". In: *Neural Computing Research Group Report* (1994).

[21] Erdem Bıyık, Malayandi Palan, Nicholas C Landolfi, Dylan P Losey, and Dorsa Sadigh. "Asking easy questions: A user-friendly approach to active reward learning". In: *arXiv preprint arXiv:1910.04365* (2019).

[22] Erik Boasson, Angelo Corsaro, and Hans van t Hag. *Eclipse Cyclone DDS*. `https://projects.eclipse.org/projects/iot.cyclonedds`.

[23] B. K. Bose. "An adaptive hysteresis-band current control technique of a voltage-fed PWM inverter for machine drive system". In: *IEEE Transactions on Industrial Electronics* 37.5 (1990).

[24] David Brandfonbrener, Stephen Tu, Avi Singh, Stefan Welker, Chad Boodoo, Nikolai Matni, and Jake Varley. "Visual Backtracking Teleoperation: A Data Collection Protocol for Offline Image-Based Reinforcement Learning". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11336–11342.

[25] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[26] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *Conference on Robot Learning (CoRL)* (2023).

[27] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. "RT-1: Robotics transformer for real-world control at scale". In: *Robotics: Science and Systems (RSS)* (2023).

[28] Alan Brown. "Robin deals with a world where things are changing all around it". In: *Amazon Science Blog* (Apr. 2022). URL: `https://www.amazon.science/latest-news/robin-deals-with-a-world-where-things-are-changing-all-around-it`.

[29] Daniel S Brown, Wonjoon Goo, and Scott Niekum. "Better-than-Demonstrator Imitation Learning via Automaticaly-Ranked Demonstrations". In: *Conference on Robot Learning (CoRL)*. 2019.

[30] Daniel S Brown, Scott Niekum, Russell Coleman, and Ravi Srinivasan. "Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences". In: *International Conference on Machine Learning*. 2020.

[31] Daniel S. Brown, Wonjoon Goo, Nagarajan Prabhat, and Scott Niekum. "Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*. 2019.

[32] Maya Cakmak, Crystal Chao, and Andrea L Thomaz. "Designing interactions for robot active learners". In: *IEEE Transactions on Autonomous Mental Development* 2.2 (2010), pp. 108–118.

[33] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set". In: *IEEE Robotics & Automation Magazine* 22.3 (Sept. 2015), pp. 36–52.

[34] John Canny. "A computational approach to edge detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1986), pp. 679–698.

[35] Claire Chen, Krishna Parasuram Srinivasan, Jeffrey O. Zhang, and Junwu Zhang. "Dexterous Manipulation Primitives for the Real Robot Challenge". In: *ArXiv preprint arXiv:2101.11597* (2021).

[36] Jessie YC Chen and Michael J Barnes. "Human–agent teaming for multirobot control: A review of human factors issues". In: *IEEE Transactions on Human-Machine Systems* 44.1 (2014), pp. 13–29.

[37] Jessie YC Chen and Michael J Barnes. "Supervisory control of multiple robots: Effects of imperfect automation and individual differences". In: *Human Factors* 54.2 (2012), pp. 157–174.

[38] Jessie YC Chen, Michael J Barnes, and Michelle Harper-Sciarini. "Supervisory control of multiple robots: Human-performance issues and user-interface design". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41.4 (2010), pp. 435–454.

[39] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. "Deep Imitation Learning for Autonomous Driving in Generic Urban Scenarios with Enhanced Safety". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 2884–2890.

[40] Kaiyuan Chen, Jiachen Yuan, Nikhil Jha, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. "FogROS G: Enabling Secure, Connected and Mobile Fog Robotics with Global Addressability". In: *arXiv preprint arXiv:2210.11691* (2022).

[41] Kaiyuan Eric Chen, Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, and Ken Goldberg. "FogROS: An Adaptive Framework for Automating Fog Robotics Deployment". In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2021, pp. 2035–2042.

[42] Sonia Chernova and Manuela Veloso. "Interactive policy learning through confidence-based autonomy". In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 1–25.

[43] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *arXiv preprint arXiv:2303.04137* (2023).

[44] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. "Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots". In: *arXiv preprint arXiv: 2402.10329*. 2024.

[45] Shih-Yi Chien, Michael Lewis, Siddharth Mehrotra, and Katia Sycara. "Imperfect automation in scheduling operator attention on control of multi-robots". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 57. 1. SAGE Publications Sage CA: Los Angeles, CA. 2013, pp. 1169–1173.

[46] Shih-Yi Chien, Yi-Ling Lin, Pei-Ju Lee, Shuguang Han, Michael Lewis, and Katia Sycara. "Attention allocation for human multi-robot control: Cognitive analysis based on behavior data and hidden states". In: *International Journal of Human-Computer Studies* 117 (2018), pp. 30–44.

[47] Susan E. F. Chipman. *The Oxford Handbook of Cognitive Science*. Oxford University Press, Oct. 2017. ISBN: 9780199842193.

[48] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. "Learning to gather information via imitation". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 908–915.

[49] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep reinforcement learning from human preferences". In: *Proc. Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[50] Eric Chu. "How Zoox Builds Autonomous Vehicles from the Wheels Up-Blog". In: *AI Exchange* (Apr. 2022). URL: `https://exchange.scale.com/public/blogs/how-zoox-builds-autonomous-vehicles-from-the-wheels-up` (visited on 06/16/2022).

[51] Kurtland Chua, Roberto Calandra, Rowan Thomas McAllister, and Sergey Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Neural Information Processing Systems*. 2018.

[52] Geoffrey Cideron, Baruch Tabanpour, Sebastian Curi, Sertan Girgin, Leonard Hussenot, Gabriel Dulac-Arnold, Matthieu Geist, Olivier Pietquin, and Robert Dadashi. "Get Back Here: Robust Imitation by Return-to-Distribution Planning". In: *arXiv preprint arXiv:2305.01400* (2023).

[53] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. "End-to-end driving via conditional imitation learning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4693–4700.

[54] Felipe Codevilla, Eder Santana, Lopez Antonio M., and Adrien Gaidon. "Exploring the Limitations of Behavior Cloning for Autonomous Driving". In: *International Conference on Computer Vision* (2019).

[55] John James Collins, David Howard, and J. Leitner. "Quantifying the Reality Gap in Robotic Manipulation Tasks". In: *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 6706–6712.

[56] Robert B Cooper. "Queueing theory". In: *Proceedings of the ACM'81 conference*. 1981, pp. 119–122.

[57] Jacob W Crandall, Mary L Cummings, Mauro Della Penna, and Paul MA De Jong. "Computing the effects of operator attention allocation in human control of multiple robots". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 41.3 (2010), pp. 385–397.

[58] Jacob W Crandall, Michael A Goodrich, Dan R Olsen, and Curtis W Nielsen. "Validating human-robot interaction schemes in multitasking environments". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 35.4 (2005), pp. 438–449.

[59] C. Crick, G. Jay, S. Osentoski, and O. C. Jenkins. "ROS and Rosbridge: Roboticists out of the loop". In: *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2012, pp. 493–494.

[60] Yuchen Cui, Siddharth Karamcheti, Raj Palleti, Nidhya Shivakumar, Percy Liang, and Dorsa Sadigh. "No, to the Right: Online Language Corrections for Robotic Manipulation via Shared Autonomy". In: *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction* (2023).

[61] Joan Daemen and Vincent Rijmen. "AES proposal: Rijndael". In: (1999).

[62] Abhinav Dahiya, Nima Akbarzadeh, Aditya Mahajan, and Stephen L Smith. "Scalable operator allocation for multi-robot assistance: A restless bandit approach". In: *IEEE Transactions on Control of Network Systems* (2022).

[63] Murtaza Dalal, Ajay Mandlekar, Caelan Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. "Imitating Task and Motion Planning with Visuomotor Transformers". In: *Conference on Robot Learning (CoRL)* (2023).

[64] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. "RoboNet: Large-scale multi-robot learning". In: *Conference on Robot Learning (CoRL)*. Vol. 100. PMLR. 2019, pp. 885–897.

[65] Andrew Daw, Robert C Hampshire, and Jamol Pender. "How to Staff when Customers Arrive in Batches". In: *arXiv e-prints* (2019), arXiv–1907.

[66] 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*. 2009, pp. 248–255.

[67] M Bernardine Dias, Balajee Kannan, Brett Browning, E Jones, Brenna Argall, M Freddie Dias, Marc Zinck, M Veloso, and Anthony Stentz. "Sliding autonomy for peer-to-peer human-robot teams". In: *Proceedings of the international conference on intelligent autonomous systems*. 2008, pp. 332–341.

[68] Andreas Doumanoglou, Jan Stria, Georgia Peleka, Ioannis Mariolis, Vladimír Petrík, Andreas Kargakos, Libor Wagner, Václav Hlaváč, Tae-Kyun Kim, and Sotiris Malassiotis. "Folding Clothes Autonomously: A Complete Pipeline". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1461–1478.

[69] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. "Bridge data: Boosting generalization of robotic skills with cross-domain datasets". In: *arXiv preprint arXiv:2109.13396* (2021).

[70] Arthur Stanley Eddington. *The mathematical theory of relativity*. The University Press, 1923.

[71] eProsima. *Fast DDS*. `https://www.eprosima.com/index.php/products-all/eprosima-fast-dds`.

[72] Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. "Adversarial Motion Priors Make Good Substitutes for Complex Reward Functions". In: *IEEE/RSJ International Conference on Robots and Systems (IROS)* (2022).

[73] Bin Fang, Shidong Jia, Di Guo, Muhua Xu, Shuhuan Wen, and Fuchun Sun. "Survey of imitation learning for robotic manipulation". In: *International Journal of Intelligent Robotics and Applications* 3.4 (2019), pp. 362–369.

[74] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Visual Imitation Learning via Meta-Learning". In: *Conf. on Robot Learning (CoRL)* (2017).

[75] Peter R. Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian S. Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. "Implicit Behavioral Cloning". In: *Conference on Robot Learning (CoRL)*. 2021.

[76] *FogROS2 Official ROS2 Repository*. `https://index.ros.org/p/fogros2/`.

[77] Justin Fu, Katie Luo, and Sergey Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *arXiv preprint arXiv:1710.11248* (2017).

[78] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proc. Int. Conf. on Machine Learning*. 2018.

[79] Niklas Funk, Charles Schaff, Rishabh Madan, Takuma Yoneda, Julen Urain De Jesus, Joe Watson, Ethan K. Gordon, Felix Widmaier, Stefan Bauer, Siddhartha S. Srinivasa, Tapomayukh Bhattacharjee, Matthew R. Walter, and Jan Peters. "Benchmarking Structured Policies and Policy Optimization for Real-World Dexterous Object Manipulation". In: *IEEE Robotics and Automation Letters* 7.1 (Jan. 2022), pp. 478–485.

[80] Aditya Ganapathi, Priya Sundaresan, Brijen Thananjeyan, Ashwin Balakrishna, Daniel Seita, Jennifer Grannen, Minho Hwang, Ryan Hoque, Joseph E Gonzalez, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. "Learning Dense Visual Correspondences in Simulation to Smooth and Fold Real Fabrics". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2021.

[81] Kanishk Gandhi, Siddharth Karamcheti, Madeline Liao, and Dorsa Sadigh. "Eliciting Compatible Demonstrations for Multi-Human Imitation Learning". In: *Conference on Robot Learning (CoRL)*. 2022.

[82] Irene Garcia-Camacho, Martina Lippi, Michael C. Welle, Hang Yin, Rika Antonova, Anastasiia Varava, Julia Borras, Carme Torras, Alessandro Marino, Guillem Alenyà, and Danica Kragic. "Benchmarking Bimanual Cloth Manipulation". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1111–1118.

[83] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley. "Desktop teleoperation via the World Wide Web". In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 1. 1995, pp. 654–659.

[84] Matthew Gombolay, Ronald Wilcox, and Julie Shah. "Fast scheduling of multi-robot teams with temporospatial constraints". In: (2013).

[85] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. 2014.

[86] *Google Remote Procedure Call*. `https://grpc.io/`.

[87] Abhishek Gupta, Justin Yu, Tony Z. Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. "Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention". In: *arXiv preprint arXiv:2104.11203* (2021).

[88] Huy Ha and Shuran Song. "FlingBot: The Unreasonable Effectiveness of Dynamic Manipulation for Cloth Unfolding". In: *Conference on Robot Learning (CoRL)*. 2021.

[89] Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)*. 2018.

[90] Sami Salama Hussen Hajjaj and Khairul Saleh Mohamed Sahari. "Establishing remote networks for ROS applications via Port Forwarding: A detailed tutorial". In: *International Journal of Advanced Robotic Systems* 14.3 (2017).

[91] Siddhant Haldar, Jyothish Pari, Anant Rai, and Lerrel Pinto. "Teach a Robot to FISH: Versatile Imitation from One Minute of Demonstrations". In: *arXiv preprint arXiv:2303.01497* (2023).

[92] Ankur Handa, Karl Van Wyk, Wei Yang, Jacky Liang, Yu-Wei Chao, Qian Wan, Stan Birchfield, Nathan D. Ratliff, and Dieter Fox. "DexPilot: Vision-Based Teleoperation of Dexterous Robotic Hand-Arm System". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 9164–9170.

[93] Sandra G Hart. "NASA-task load index (NASA-TLX); 20 years later". In: *Proceedings of the human factors and ergonomics society annual meeting*. Vol. 50. Sage publications Sage CA: Los Angeles, CA. 2006, pp. 904–908.

[94] Alexander Herzog, Kanishka Rao, Karol Hausman, Yao Lu, Paul Wohlhart, Mengyuan Yan, Jessica Lin, Montse Gonzalez Arenas, Ted Xiao, Daniel Kappler, Daniel Ho, Jarek Rettinghouse, Yevgen Chebotar, Kuang-Huei Lee, Keerthana Gopalakrishnan, Ryan C. Julian, Adrian Li, Chuyuan Fu, Bo Wei, Sangeetha Sukumari Ramesh, K. D. Holden, Kim Kleiven, David Rendleman, Sean Kirmani, Jeffrey Bingham, Jonathan Weisz, Ying Xu, Wenlong Lu, Matthew Bennice, Cody Fong, David Do, Jessica Lam, Yunfei Bai, Benjie Holson, Michael J. Quinlan, Noah Brown, Mrinal Kalakrishnan, Julian Ibarz, Peter Pastor, and Sergey Levine. "Deep RL at Scale: Sorting Waste in Office Buildings with a Fleet of Mobile Manipulators". In: *Robotics: Science and Systems (RSS)*. 2023.

[95] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *Advances in Neural Information Processing Systems*. 2016.

[96] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Neural Information Processing Systems (NeurIPS)* (2020).

[97] Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S Brown, and Ken Goldberg. "ThriftyDAgger: Budget-aware novelty and risk gating for interactive imitation learning". In: *Conference on Robot Learning (CoRL)* (2021).

[98] Ryan Hoque, Ashwin Balakrishna, Carl Putterman, Michael Luo, Daniel S. Brown, Daniel Seita, Brijen Thananjeyan, Ellen Novoseller, and Ken Goldberg. "LazyDAgger: Reducing context switching in interactive imitation learning". In: *International Conference on Automation Sciences and Engineering (CASE)*. 2021.

[99] Ryan Hoque, Lawrence Y. Chen, Satvik Sharma, Karthik Dharmarajan, Brijen Thananjeyan, Pieter Abbeel, and Ken Goldberg. "Fleet-DAgger: Interactive Robot Fleet Learning with Scalable Human Supervision". In: *Conference on Robot Learning (CoRL)*. 2022.

[100] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. "VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation". In: *Proc. Robotics: Science and Systems (RSS)*. 2020.

[101] Ryan Hoque, Kaushik Shivakumar, Shrey Aeron, Gabriel Deza, Aditya Ganapathi, Adrian Wong, Johnny Lee, Andy Zeng, Vincent Vanhoucke, and Ken Goldberg. "Learning to Fold Real Garments with One Arm: A Case Study in Cloud-Based Robotics Research". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022.

[102] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. "Reward learning from human preferences and demonstrations in Atari". In: *Advances in Neural Information Processing Systems*. 2018.

[103] Jeffrey Ichnowski and Ron Alterovitz. "Motion Planning Templates: A Motion Planning Framework for Robots with Low-power CPUs". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[104] Jeffrey Ichnowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Vıctor Mayoral-Vilches, Hugo Zhan, Derek Xu, Ramtin Ghassemi, John Kubiatowicz, et al. "FogROS2: An Adaptive and Extensible Platform for Cloud and Fog Robotics Using ROS 2". In: *arXiv preprint arXiv:2205.09778* (2022).

[105] Jeffrey Ichnowski, William Lee, Victor Murta, Samuel Paradis, Ron Alterovitz, Joseph E Gonzalez, Ion Stoica, and Ken Goldberg. "Fog Robotics Algorithms for Distributed Motion Planning Using Lambda Serverless Computing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4232–4238.

[106] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. "Dynamical movement primitives: learning attractor models for motor behaviors". In: *Neural computation* 25.2 (2013).

[107] *Integrating ROS2 with Eclipse zenoh*. `https://zenoh.io/blog/2021-04-28-ros2-integration/`. Accessed: 2021-02-15.

[108] Alex Irpan. *Deep Reinforcement Learning Doesn't Work Yet*. `https://www.alexirpan.com/2018/02/14/rl-hard.html`. 2018.

[109] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *Conference on Robot Learning*. 2021.

[110] Snehal Jauhri, Carlos Celemin, and Jens Kober. "Interactive Imitation Learning in State-Space". In: *arXiv preprint arXiv:2008.00524* (2020).

[111] Harold Jeffreys. *The Theory of Probability*. Oxford University Press, 1939.

[112] Tianchen Ji, Roy Dong, and Katherine Driggs-Campbell. "Traversing Supervisor Problem: An Approximately Optimal Approach to Multi-Robot Assistance". In: *Robotics: Science and Systems (RSS)* (2022).

[113] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. "VIMA: General Robot Manipulation with Multimodal Prompts". In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022.

[114] Edward Johns. "Coarse-to-Fine Imitation Learning: Robot Manipulation from a Single Demonstration". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 4613–4619.

[115] Kshitij Judah, Alan Fern, and Thomas Dietterich. "Active imitation learning via state queries". In: *Proceedings of the icml workshop on combining learning strategies to reduce label cost*. Citeseer. 2011.

[116] Gregory Kahn, Pieter Abbeel, and Sergey Levine. "LaND: Learning to Navigate from Disengagements". In: *arXiv preprint arXiv:2010.04689*. 2020.

[117] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Conference on Robot Learning (CoRL)* (2018).

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

[119] Peter Kazanzides, Zihan Chen, Anton Deguet, Gregory S Fischer, Russell H Taylor, and Simon P DiMaio. "An open-source research kit for the da Vinci® Surgical System". In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 6434–6439.

[120] Ben Kehoe, Dmitry Berenson, and Ken Goldberg. "Estimating part tolerance bounds based on adaptive cloud-based grasp planning with slip". In: *IEEE Conference on Automation Science and Engineering (CASE)*. 2012, pp. 1106–1113.

[121] Ben Kehoe, Dmitry Berenson, and Ken Goldberg. "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2012, pp. 576–583.

[122] Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. "Cloud-based robot grasping with the google object recognition engine". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2013, pp. 4263–4270.

[123] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. "A survey of research on cloud robotics and automation". In: *IEEE Trans. Automation Science and Engineering* 12.2 (2015), pp. 398–409.

[124] Ben Kehoe, Deepak Warrier, Sachin Patil, and Ken Goldberg. "Cloud-based grasp analysis and planning for toleranced parts using parallelized Monte Carlo sampling". In: *IEEE Trans. Automation Science and Engineering* 12.2 (2014), pp. 455–470.

[125] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. "HG-DAgger: Interactive Imitation Learning with Human Experts". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2019.

[126] Taylor Kessler Faulkner, Reymundo A Gutierrez, Elaine Schaertl Short, Guy Hoffman, and Andrea L Thomaz. "Active attention-modified policy shaping: socially interactive agents track". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019.

[127] Beomjoon Kim and Joelle Pineau. "Maximum Mean Discrepancy Imitation Learning." In: *Robotics: Science and systems*. 2013.

[128] Chung Min Kim, Michael Danielczuk, Isabella Huang, and Ken Goldberg. "Simulation of Parallel-Jaw Grasping using Incremental Potential Contact Models". In: *arXiv preprint arXiv:2111.01391* (2021).

[129] Ji Woong Kim, Peiyao Zhang, Peter L. Gehlbach, Iulian I. Iordachita, and Marin Kobilarov. "Towards Autonomous Eye Surgery by Combining Deep Imitation Learning with Optimal Control". In: *Conference on Robot Learning (CoRL)*. 2020.

[130] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2014.

[131] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. "Federated Learning: Strategies for Improving Communication Efficiency". In: *NIPS Workshop on Private Multi-Party Machine Learning*. 2016.

[132] David Kortenkamp, Debra Keirn-Schreckenghost, and R Peter Bonasso. "Adjustable control autonomy for manned space flight". In: *2000 IEEE Aerospace Conference. Proceedings (Cat. No. 00TH8484)*. Vol. 7. IEEE. 2000, pp. 629–640.

[133] Oliver Kroemer, Scott Niekum, and George Konidaris. "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms." In: *Journal of Machine Learning Research* 22 (2021), pp. 30–1.

[134] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. "RMA: Rapid Motor Adaptation for Legged Robots". In: *Robotics: Science and Systems (RSS)*. 2021.

[135] Andrey Kurenkov, Ajay Mandlekar, Roberto Martin-Martin, Silvio Savarese, and Animesh Garg. "AC-Teach: A Bayesian Actor-Critic Method for Policy Learning with an Ensemble of Suboptimal Teachers". In: *Conf. on Robot Learning (CoRL)*. 2019.

[136] Miu-Ling Lam and Kit-Yung Lam. "Path planning as a service PPaaS: Cloud-based robotic path planning". In: *Proc. IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*. 2014, pp. 1839–1844.

[137] Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations". In: *International Conference on Robotics and Automation (ICRA)*. 2017, pp. 358–365.

[138] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. "DART: Noise Injection for Robust Imitation Learning". In: *Conf. on Robot Learning (CoRL)*. 2017.

[139] Michael Laskey, Sam Staszak, Wesley Hsieh, Jeffrey Mahler, Florian Pokorny, Anca Dragan, and Ken Goldberg. "SHIV: Reducing Supervisor Burden using Support Vectors for Efficient Learning from Demonstrations in High Dimensional State Spaces". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2016.

[140] John Launchbury. "A Natural Semantics for Lazy Evaluation". In: New York, NY, USA: Association for Computing Machinery, 1993, pp. 144–154. ISBN: 0897915607.

[141] Yann LeCun, Sumit Chopra, Raia Hadsell, Aurelio Ranzato, and Fu Jie Huang. "A Tutorial on Energy-Based Learning". In: *Predicting Structured Data* 1.0 (2006).

[142] Michael Lewis. "Human interaction with multiple remote robots". In: *Reviews of Human Factors and Ergonomics* 9.1 (2013), pp. 131–174.

[143] Pusong Li, Bill DeRose, Jeffrey Mahler, Juan Aparicio Ojea, Ajay Kumar Tanwani, and Ken Goldberg. "Dex-Net as a service (DNaaS): A cloud-based robust robot grasp planning system". In: *IEEE Conference on Automation Science and Engineering (CASE)*. 2018, pp. 1420–1427.

[144] Jia Zhi Lim and Danny Wee-Kiat Ng. "Cloud based implementation of ROS through VPN". In: *Int. Conf. on Smart Computing & Communications (ICSCC)*. IEEE. 2019, pp. 1–5.

[145] J. Lin. "Divergence measures based on the Shannon entropy". In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 145–151.

[146] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. "SoftGym: Benchmarking Deep Reinforcement Learning for Deformable Object Manipulation". In: *Conference on Robot Learning (CoRL)* (2020).

[147] Huihan Liu, Shivin Dass, Roberto Mart'in-Mart'in, and Yuke Zhu. "Model-Based Runtime Monitoring with Interactive Imitation Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024.

[148] Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. "Robot Learning on the Job: Human-in-the-Loop Autonomy and Learning During Deployment". In: *arXiv* abs/2211.08416 (2022).

[149] "Logistics Automation with Plus One Robotics". en-US. In: *Parcel Monitor* (May 2022). URL: `https://www.parcelmonitor.com/blog/tech-spotlight-logistics-automation-with-plus-one-automation/` (visited on 06/16/2022).

[150] Jianlan Luo, Perry Dong, Yuexiang Zhai, Yi Ma, and Sergey Levine. "RLIF: Interactive Imitation Learning as Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2024.

[151] Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Wenzhao Lian, Chang Su, Mel Vecerik, Ning Ye, Stefan Schaal, and Jon Scholz. "Robust multi-modal policies for industrial assembly via reinforcement learning and demonstrations: A large-scale study". In: *arXiv preprint arXiv:2103.11512* (2021).

[152] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. "Interactive language: Talking to robots in real time". In: *IEEE Robotics and Automation Letters* (2023).

[153] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605.

[154] Alexis C. Madrigal. "Waymo's Robot Cars, and the Humans Who Tend to Them". en. In: *The Atlantic* (Aug. 2018). Section: Technology. URL: `https://www.theatlantic.com/technology/archive/2018/08/waymos-robot-cars-and-the-humans-who-tend-to-them/568051/` (visited on 06/16/2022).

[155] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In: *Robotics: Science and Systems (RSS)*. 2017.

[156] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 2308–2315.

[157] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning". In: *arXiv preprint arXiv:2108.10470* (2021).

[158] Ajay Mandlekar, Jonathan Booher, Max Spero, Albert Tung, Anchit Gupta, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. "Scaling Robot Supervision to Hundreds of Hours with RoboTurk: Robotic Manipulation Dataset through Human Reasoning and Dexterity". In: *arXiv preprint arXiv:1911.04052* (2019).

[159] Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. "MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations". In: *Conference on Robot Learning (CoRL)*. 2023.

[160] Ajay Mandlekar, Danfei Xu, Roberto Martin-Martin, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. "Human-in-the-Loop Imitation Learning using Remote Teleoperation". In: *ArXiv preprint arXiv:2012.06733* (2020).

[161] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. "Learning to Generalize Across Long-Horizon Tasks from Human Demonstrations". In: *arXiv preprint arXiv:2003.06085* (2020).

[162] Ajay Mandlekar, Danfei Xu, J. Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martin-Martin. "What Matters in Learning from Offline Human Demonstrations for Robot Manipulation". In: *Conference on Robot Learning (CoRL)*. 2021.

[163] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. "RoboTurk: A Crowdsourcing Platform for Robotic Skill Learning through Imitation". In: *Conference on Robot Learning*. 2018.

[164] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. "Adversarially robust policy learning: Active construction of physically-plausible perturbations". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3932–3939.

[165] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. "Rapid Locomotion via Reinforcement Learning". In: *arXiv preprint arXiv:2205.02824* (2022).

[166] Nicholas D Matsakis and Felix S Klock II. "The rust language". In: *ACM SIGAda Ada Letters*. Vol. 34. 3. ACM. 2014, pp. 103–104.

[167] Vıctor Mayoral-Vilches, Ruffin White, Gianluca Caiazza, and Mikael Arguedas. "SROS2: Usable Cyber Security Tools for ROS 2". In: *arXiv e-prints* (2022), arXiv–2208.

[168] Robert McCarthy, Francisco Roldan Sanchez, Qiang Wang, David Cordova Bulens, Kevin McGuinness, Noel O'Connor, and Stephen J. Redmond. "Solving the Real Robot Challenge using Deep Reinforcement Learning". In: *arXiv preprint arXiv: 2109.15233* (2021).

[169] Michael James McDonald and Dylan Hadfield-Menell. "Guided imitation of task and motion planning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 630–640.

[170] Garrett McGrath and Paul R Brenner. "Serverless computing: Design, implementation, and performance". In: *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE. 2017, pp. 405–410.

[171] Kunal Menda, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. "EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2019.

[172] Suvir Mirchandani, F. Xia, Peter R. Florence, Brian Ichter, Danny Driess, Montse Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. "Large Language Models as General Pattern Machines". In: *Conference on Robot Learning (CoRL)*. 2023.

[173] Nitesh Mor, Richard Pratt, Eric Allman, Kenneth Lutz, and John Kubiatowicz. "Global data plane: A federated vision for secure data in edge computing". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 1652–1663.

[174] Douglas Morrison, Peter Corke, and J. Leitner. "EGAD! An Evolved Grasping Analysis Dataset for Diversity and Reproducibility in Robotic Manipulation". In: *IEEE Robotics and Automation Letters* 5 (2020), pp. 4368–4375.

[175] *Multicast DNS RFC 6762*. `https://www.rfc-editor.org/rfc/rfc6762.html`.

[176] Raul Mur-Artal and Juan D Tardós. "ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras". In: *IEEE Trans. Robotics* 33.5 (2017), pp. 1255–1262.

[177] Robin R Murphy and Erika Rogers. "Cooperative assistance for remote robot supervision". In: *Presence: Teleoperators & Virtual Environments* 5.2 (1996), pp. 224–240.

[178] Michael Myers, Carlisle Adams, Dave Solo, and David Kemp. *Internet X. 509 certificate request message format*. Tech. rep. 1999.

[179] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, P. Abbeel, Jitendra Malik, and Sergey Levine. "Combining self-supervised learning and imitation for vision-based rope manipulation". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 2146–2153.

[180] Radford M Neal. "Annealed importance sampling". In: *Statistics and computing* 11 (2001), pp. 125–139.

[181] Hai Nguyen, Andrea Baisero, Dian Wang, Christopher Amato, and Robert Platt. "Leveraging fully observable policies for learning under partial observability". In: *arXiv preprint arXiv:2211.01991* (2022).

[182] Frank Nielsen. "Fast Approximations of the Jeffreys Divergence between Univariate Gaussian Mixtures via Mixture Conversions to Exponential-Polynomial Distributions". In: *Entropy* 23 (2021).

[183] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. "Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks". In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[184] Dan R Olsen Jr and Stephen Bart Wood. "Fan-out: Measuring human control of multiple robots". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2004, pp. 231–238.

[185] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding". In: *ArXiv preprint arXiv:1807.03748* (2018).

[186] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: `2303.08774 [cs.CL]`.

[187] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[188] Claudio Pacchierotti, Stephen Sinclair, Massimiliano Solazzi, Antonio Frisoli, Vincent Hayward, and Domenico Prattichizzo. "Wearable Haptic Systems for the Fingertip and the Hand: Taxonomy, Review, and Perspectives". In: *IEEE Transactions on Haptics* 10.4 (2017), pp. 580–600.

[189] Abhishek Padalkar et al. "Open X-Embodiment: Robotic Learning Datasets and RT-X Models". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024.

[190] Malayandi Palan, Nicholas C. Landolfi, Gleb Shevchuk, and Dorsa Sadigh. "Learning Reward Functions by Integrating Human Demonstrations and Preferences". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2019.

[191] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. "Agile Autonomous Driving using End-to-End Deep Imitation Learning". In: *Proc. Robotics: Science and Systems (RSS)*. 2018.

[192] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. "NetBricks: Taking the V out of NFV". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 203–216.

[193] Samuel Paradis, Minho Hwang, Brijen Thananjeyan, Jeffrey Ichnowski, Daniel Seita, Danyal Fer, Thomas Low, Joseph E. Gonzalez, and Ken Goldberg. "Intermittent Visual Servoing: Efficiently Learning Policies Robust to Instrument Changes for High-precision Surgical Manipulation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 7166–7173.

[194] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. "Probabilistic movement primitives". In: *Advances in neural information processing systems*. 2013, pp. 2616–2624.

[195] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. "Duckietown: An open, inexpensive and flexible platform for autonomy education and research". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1497–1504.

[196] Andi Peng, Aviv Netanyahu, Mark K Ho, Tianmin Shu, Andreea Bobu, Julie Shah, and Pulkit Agrawal. "Diagnosis, Feedback, Adaptation: A Human-in-the-Loop Framework for Test-Time Policy Adaptation". In: *International Conference on Machine Learning* (2023).

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

[198] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. "ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters". In: *arXiv preprint arXiv:2205.01906* (2022).

[199] Alyson Benoni Matias Pereira, Ricardo Emerson Julio, and Guilherme Sousa Bastos. "ROSRemote: Using ROS on Cloud to Access Robots Remotely". In: *Robot Operating System (ROS)*. Springer, 2019, pp. 569–605.

[200] Jeffrey R Peters, Vaibhav Srivastava, Grant S Taylor, Amit Surana, Miguel P Eckstein, and Francesco Bullo. "Human supervisory control of robotic teams: Integrating cognitive modeling with engineering design". In: *IEEE Control Systems Magazine* 35.6 (2015), pp. 57–80.

[201] *Piab piSOFTGRIP Gripper*. 2022. URL: `https://www.piab.com/en-us/suction-cups-and-soft-grippers/soft-grippers/pisoftgrip-vacuum-driven-soft-gripper-/pisoftgrip-/#overview`.

[202] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. "The Robotarium: A remotely accessible swarm robotics research testbed". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1699–1706.

[203] Dean A Pomerleau. "Efficient training of artificial neural networks for autonomous navigation". In: *Neural Computation* 3.1 (1991).

[204] Dean A. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *Neural Information Processing Systems (NeurIPS)*. Ed. by D. Touretzky. Vol. 1. Morgan-Kaufmann, 1988.

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

[206] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[207] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. "Real-world humanoid locomotion with reinforcement learning". In: *Science Robotics* 9.89 (2024).

[208] Siddharth Reddy, Anca D Dragan, and Sergey Levine. "Shared autonomy via deep reinforcement learning". In: *Proc. Robotics: Science and Systems (RSS)* (2018).

[209] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. "Where Do You Think You're Going?: Inferring Beliefs about Dynamics from Behavior". In: *Proc. Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[210] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, F. Xia, Jacob Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. "Robots That Ask For Help: Uncertainty Alignment for Large Language Model Planners". In: *Conference on Robot Learning (CoRL)*. 2023.

[211] Marc Rigter, Bruno Lacerda, and Nick Hawes. "A framework for learning from demonstration with minimal human effort". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2023–2030.

[212] *rosduct*. `https://github.com/uts-magic-lab/rosduct`.

[213] Ariel Rosenfeld, Noa Agmon, Oleg Maksimov, and Sarit Kraus. "Intelligent agent supporting human–multi-robot team collaboration". In: *Artificial Intelligence* 252 (2017), pp. 211–231.

[214] Stephane Ross and J. Andrew Bagnell. *Reinforcement and Imitation Learning via Interactive No-Regret Learning*. 2014. arXiv: `1406.5979 [cs.LG]`.

[215] Stephane Ross, Geoffrey J Gordon, and J Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

[216] William B Rouse. "Adaptive allocation of decision making responsibility between supervisor and computer". In: *Monitoring behavior and supervisory control*. Springer, 1976, pp. 295–306.

[217] *RSASSA-PSS RFC 4056*. `https://www.rfc-editor.org/rfc/rfc4056`.

[218] *RTI Connext DDS*. `https://www.rti.com/products`.

[219] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[220] Dorsa Sadigh, Anca D. Dragan, S. Shankar Sastry, and Sanjit A. Seshia. "Active Preference-Based Learning of Reward Functions". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2017.

[221] Akanksha Saran, Elaine Schaertl Short, Andrea Thomaz, and Scott Niekum. "Understanding Teacher Gaze Patterns for Robot Learning". In: ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, Oct. 2020, pp. 1247–1258.

[222] William Saunders, Girish Sastry, Andreas Stuhlmüller, and Owain Evans. "Trial without Error: Towards Safe RL with Human Intervention". In: *17th International Conference on Autonomous Agents and MultiAgent Systems* (2018).

[223] *Scalable Distributed Robot Fleet With Fast DDS Discovery Server*. `https://husarnet.com/blog/ros2-dds-discovery-server`. Accessed: 2023-03-1.

[224] Paul Scerri, David V Pynadath, and Milind Tambe. "Towards adjustable autonomy for the real world". In: *Journal of Artificial Intelligence Research* 17 (2002), pp. 171–228.

[225] John Schulman, Sergey Levine, P. Abbeel, Michael I. Jordan, and Philipp Moritz. "Trust Region Policy Optimization". In: *International Conference on Machine Learning*. 2015.

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

[227] Daniel Seita, Aditya Ganapathi, Ryan Hoque, Minho Hwang, Edward Cen, Ajay Kumar Tanwani, Ashwin Balakrishna, Brijen Thananjeyan, Jeffrey Ichnowski, Nawid Jamali, Katsu Yamane, Soshi Iba, John Canny, and Ken Goldberg. "Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2020.

[228] Brennan Sellner, Frederik W Heger, Laura M Hiatt, Reid Simmons, and Sanjiv Singh. "Coordinated multiagent teams and sliding autonomy for large-scale assembly". In: *Proceedings of the IEEE* 94.7 (2006), pp. 1425–1444.

[229] Brennan Sellner, Reid Simmons, and Sanjiv Singh. "User modelling for principled sliding autonomy in human-robot teams". In: *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 2005, pp. 197–208.

[230] Lui Sha, Tarek Abdelzaher, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, Aloysius K Mok, et al. "Real time scheduling theory: A historical perspective". In: *Real-time systems* 28.2 (2004), pp. 101–155.

[231] Nur Muhammad Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. "Behavior Transformers: Cloning k modes with one stone". In: *Neural Information Processing Systems (NeurIPS)*. 2022.

[232] Claude Shannon. "A mathematical theory of communication". In: *Bell System Technical Journal (1948)* (1948).

[233] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), pp. 640–651.

[234] Thomas B Sheridan. "Adaptive automation, level of automation, allocation authority, supervisory control, and adaptive control: Distinctions and modes of adaptation". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 41.4 (2011), pp. 662–667.

[235] Lucy Shi, Archit Sharma, Tony Zhao, and Chelsea Finn. "Waypoint-Based Imitation Learning for Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. 2023.

[236] Lucy Xiaoyang Shi, Zheyuan Hu, Tony Zhao, Archit Sharma, Karl Pertsch, Jianlan Luo, Sergey Levine, and Chelsea Finn. "Yell At Your Robot: Improving On-the-Fly from Language Corrections". In: *ArXiv preprint arXiv:2403.12910* (2024).

[237] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. 2022.

[238] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. "ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 10737–10746.

[239] *Signature and hash algorithms for TLS and DTLS*. `https://www.ibm.com/docs/en/zos/2.5.0?topic=support-signature-hash-algorithms`.

[240] Jennifer Smith. "Robotic Arms Are Using Machine Learning to Reach Deeper Into Distribution". en-US. In: *Wall Street Journal* (Jan. 2022). ISSN: 0099-9660. URL: `https://tinyurl.com/2p89zb35` (visited on 06/16/2022).

[241] Jonathan Spencer, Sanjiban Choudhury, Matthew Barnes, Matthew Schmittle, Mung Chiang, Peter Ramadge, and Siddhartha Srinivasa. "Learning from Interventions: Human-robot Interaction as both Explicit and Immplicit Feedback". In: *Proc. Robotics: Science and Systems (RSS)*. 2020.

[242] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. "OpenVSLAM: A Versatile Visual SLAM Framework". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: ACM, 2019, pp. 2292–2295.

[243] Xiatao Sun, Shuo Yang, and Rahul Mangharam. "MEGA-DAgger: Imitation Learning with Multiple Imperfect Experts". In: *ArXiv* arXiv preprint arXiv:2303.00638 (2023).

[244] Priya Sundaresan, Priya Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E. Gonzalez, and Ken Goldberg. "Learning Interpretable and Transferable Rope Manipulation Policies Using Depth Sensing and Dense Object Descriptors". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020.

[245] Gokul Swamy, Siddharth Reddy, Sergey Levine, and Anca D Dragan. "Scaled autonomy: Enabling human operators to control robot fleets". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 5942–5948.

[246] Ajay Kumar Tanwani, Raghav Anand, Joseph E Gonzalez, and Ken Goldberg. "RILaaS: Robot inference and learning as a service". In: *IEEE Robotics & Automation Letters* 5.3 (2020), pp. 4423–4430.

[247] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. "Recovery rl: Safe reinforcement learning with learned recovery zones". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4915–4922.

[248] Nan Tian, Matthew Matl, Jeffrey Mahler, Yu Xiang Zhou, Samantha Staszak, Christopher Correa, Steven Zheng, Qiang Li, Robert Zhang, and Ken Goldberg. "A cloud robot system using the dexterity network and Berkeley robotics and automation as a service (BRASS)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1615–1622.

[249] Naftali Tishby and Noga Zaslavsky. "Deep learning and the information bottleneck principle". In: *2015 IEEE Information Theory Workshop (ITW)* (2015), pp. 1–5.

[250] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[251] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A Physics Engine for Model-Based Control". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2012.

[252] *Tokio*. `https://tokio.rs/`.

[253] Faraz Torabi, Garrett Warnell, and Peter Stone. "Behavioral Cloning from Observation". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. Stockholm, Sweden, July 2018.

[254] Susanne Trick, Franziska Herbert, Constantin A. Rothkopf, and Dorothea Koert. "Interactive Reinforcement Learning With Bayesian Fusion of Multimodal Advice". In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 7558–7565.

[255] Hsiao-Yu Tung, Adam W Harley, Liang-Kang Huang, and Katerina Fragkiadaki. "Reward learning from narrated demonstrations". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7004–7013.

[256] Alan Turing. "Computing Machinery and Intelligence". In: *Mind* 49 (1950), pp. 433–460.

[257] *UC Berkeley Minimum Security Standard*. `https://security.berkeley.edu/policy/minimum-security-standards-networked-devices-mssnd`.

[258] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[259] Shivam Vats, Oliver Kroemer, and Maxim Likhachev. "Synergistic Scheduling of Learning and Allocation of Tasks in Human-Robot Teams". In: *arXiv preprint arXiv: 2203.07478* (2022).

[260] Nolan Wagener, Byron Boots, and Ching-An Cheng. "Safe Reinforcement Learning Using Advantage-Based Intervention". In: *International Conference on Machine Learning (ICML)*. 2021.

[261] Fan Wang, Bo Zhou, Ke Chen, Tingxiang Fan, Xi Zhang, Jiangyong Li, Hao Tian, and Jia Pan. "Intervention Aided Reinforcement Learning for Safe and Practical Policy Optimization in Navigation". In: *Conf. on Robot Learning (CoRL)*. 2018.

[262] Max Welling and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proceedings of the 28th International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 681–688. ISBN: 9781450306195.

[263] Bowen Wen, Wenzhao Lian, Kostas E. Bekris, and Stefan Schaal. "You Only Demonstrate Once: Category-Level Manipulation from Single Visual Demonstration". In: *Robotics: Science and Systems (RSS)*. 2022.

[264] Thomas Weng, Sujay Bajracharya, Yufei Wang, Khush Agrawal, and David Held. "FabricFlowNet: Bimanual Cloth Manipulation with a Flow-based Policy". In: *Conference on Robot Learning (CoRL)*. 2021.

[265] *wireguard VPN*. `https://www.wireguard.com/`.

[266] Adrian Wong, Andy Zeng, Arnab Bose, Ayzaan Wahid, Dmitry Kalashnikov, Ivan Krasin, Jake Varley, Johnny Lee, Jonathan Tompson, Maria Attarian, Pete Florence, Robert Baruch, Sichun Xu, Stefan Welker, Vikas Sindhwani, Vincent Vanhoucke, and Wayne Gramlich. *PyReach - Python Client SDK for Robot Remote Control*. `https://github.com/google-research/pyreach`. 2022.

[267] Josiah Wong, Albert Tung, Andrey Kurenkov, Ajay Mandlekar, Li Fei-Fei, Silvio Savarese, and Roberto Martín-Martín. "Error-aware imitation learning from teleoperation data for mobile manipulation". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1367–1378.

[268] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. "Learning to Manipulate Deformable Objects without Demonstrations". In: *Robotics: Science and Systems (RSS)*. 2020.

[269] Manuel Wüthrich, Felix Widmaier, Felix Grimminger, Shruti Joshi, Vaibhav Agrawal, Bilal Hammoud, Majid Khadiv, Miroslav Bogdanovic, Vincent Berenz, Julian Viereck, Maximilien Naveau, Ludovic Righetti, Bernhard Schölkopf, and Stefan Bauer. "TriFinger: An Open-Source Robot for Learning Dexterity". In: *Conference on Robot Learning (CoRL)*. 2020.

[270] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. "Masked visual pre-training for motor control". In: *arXiv preprint arXiv:2203.06173* (2022).

[271] Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. "Learning with Training Wheels: Speeding up Training with a Simple Controller for Deep Reinforcement Learning". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2018.

[272] Binhuai Xu and Jing Bian. "A Cloud Robotic Application Platform Design Based on the Microservices Architecture". In: *Int. Conf. on Control, Robotics and Intelligent System*. 2020, pp. 13–18.

[273] Brian Yang, Jesse Zhang, Vitchyr Pong, Sergey Levine, and Dinesh Jayaraman. "REPLAB: A Reproducible Low-Cost Arm Benchmark Platform for Robotic Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[274] Sebastián A Zanlongo, Peter Dirksmeier, Philip Long, Taskin Padir, and Leonardo Bobadilla. "Scheduling and path-planning for operator oversight of multiple robots". In: *Robotics* 10.2 (2021), p. 57.

[275] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. "Transporter Networks: Rearranging the Visual World for Robotic Manipulation". In: *Conference on Robot Learning (CoRL)* (2020).

[276] Jiakai Zhang and Kyunghyun Cho. "Query-Efficient Imitation Learning for End-to-End Autonomous Driving". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2017.

[277] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Ken Goldberg, and Pieter Abbeel. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *arXiv preprint arXiv:1710.04615* (2017).

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

[279] Kuanhao Zheng, Dylan F Glas, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. "Supervisory control of multiple social robots for navigation". In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2013, pp. 17–24.

[280] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. "Robosuite: A Modular Simulation Framework and Benchmark for Robot Learning". In: *arXiv preprint arXiv:2009.12293*. 2020.

[281] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. "Maximum entropy inverse reinforcement learning." In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2008.

Part V

Appendices

Appendix A

Appendix for Chapter 2

Here we provide further details on our MuJoCo experiments, hyperparameter sensitivity, simulated fabric experiments, and physical fabric experiments.

A.1 MuJoCo

As stated in the main text, we evaluate on the HalfCheetah-v2, Walker2D-v2, and Ant-v2 environments. To train the algorithmic supervisor, we utilize the TD3 implementation from OpenAI SpinningUp (`https://spinningup.openai.com/en/latest/`) with default hyperparameters and run for 100, 200, and 500 epochs respectively. The expert policies obtain rewards of 5330.78 ± 117.65, 3492.08 ± 1110.31, and 4492.88 ± 1580.42, respectively. Note that the experts for Walker2D and Ant have high variance, resulting in higher variance for the corresponding learning curves in Figure 2.3. We provide the state space dimensionality $|S|$, action space dimensionality $|A|$, and LazyDAgger hyperparameters (see Algorithm 1) for each environment in Table A.1. The β_H value in the table is multiplied with the maximum possible action discrepancy $||a_{high} - a_{low}||_2^2$ to become the threshold for training $f(\cdot)$. In MuJoCo environments, $a_{high} = \vec{1}$ and $a_{low} = -\vec{1}$. The β_H value used for SafeDAgger in all experiments is chosen by the method provided in the paper introducing SafeDAgger [276]: the threshold at which roughly 20% of the initial offline dataset is classified as "unsafe."

For LazyDAgger and all baselines, the actor policy $\pi_R(\cdot)$ is a neural network with 2 hidden layers with 256 neurons each, rectified linear unit (ReLU) activation, and hyperbolic tangent output activation. For LazyDAgger and SafeDAgger, the discrepancy classifier $f(\cdot)$ is a neural network with 2 hidden layers with 128 neurons each, ReLU activation, and sigmoid output activation. We take 2,000 gradient steps per epoch and optimize with Adam and learning rate 1e-3 for both neural networks. To collect $\mathcal{D}$ and $\mathcal{D}_{\text{S}}$ in Algorithm 1 and SafeDAgger, we randomly partition our dataset of 4,000 state-action pairs into 70% (2,800 state-action pairs) for $\mathcal{D}$ and 30% (1,200 state-action pairs) for $\mathcal{D}_{\text{S}}$.

Environment	$\|S\|$	$\|A\|$	N	T	β_H	β_R	σ^2
HalfCheetah	16	7	10	5000	5e-3	β_H / 10	0.30
Walker2D	16	7	15	5000	5e-3	β_H / 10	0.10
Ant	111	8	15	5000	5e-3	β_H / 2	0.05

Table A.1: **MuJoCo Hyperparameters:** $|S|$ and $|A|$ are aspects of the Gym environments while the other values are hyperparameters of LazyDAgger (Algorithm 1). Note that T and β_H are the same across all environments, and that β_R is a function of β_H.

A.2 LazyDAgger Switching Thresholds

As described in Section 2.5, the main LazyDAgger hyperparameters are the safety thresholds for switching to supervisor control (β_H) and returning to autonomous control (β_R). To tune these hyperparameters in practice, we initialize β_H and β_R with the method in Zhang et al. [276]; again, this sets the safety threshold such that approximately 20% of the initial dataset is unsafe. We then tune β_H higher to balance reducing the frequency of switching to the supervisor with allowing enough supervision for high policy performance. Finally we set β_R as a multiple of β_H, starting from $\beta_R = \beta_H$ and tuning downward to balance improving the performance and increasing intervention length with keeping the total number of actions moderate. Note that since these parameters are not automatically set, we must re-run experiments for each change of parameter values. Since this tuning results in unnecessary supervisor burden, eliminating or mitigating this requirement is an important direction for future work.

To analyze sensitivity to β_R and β_H, we plot the results of a grid search over parameter values on each of the MuJoCo environments in Figure A.1. Note that a lighter color in the heatmap is more desirable for reward while a darker color is more desirable for actions and switches. We see that the supervisor burden in terms of actions and context switches is not very sensitive to the threshold as we increase β_H but jumps significantly for the very low setting ($\beta_H = 5 \times 10^{-4}$) as a large amount of data points are classified as unsafe. Similarly, we see that reward is relatively stable (note the small heatmap range for HalfCheetah) as we decrease β_H but can suffer for high values, as interventions are not requested frequently enough. Reward and supervisor burden are not as sensitive to β_R but follow the same trends we expect, with higher reward and burden as β_R decreases.

A.3 Fabric Smoothing in Simulation

Fabric Simulator

More information about the fabric simulator can be found in Seita et al. [227], but we review the salient details here. The fabric is modeled as a mass-spring system with a $n \times n$ square grid

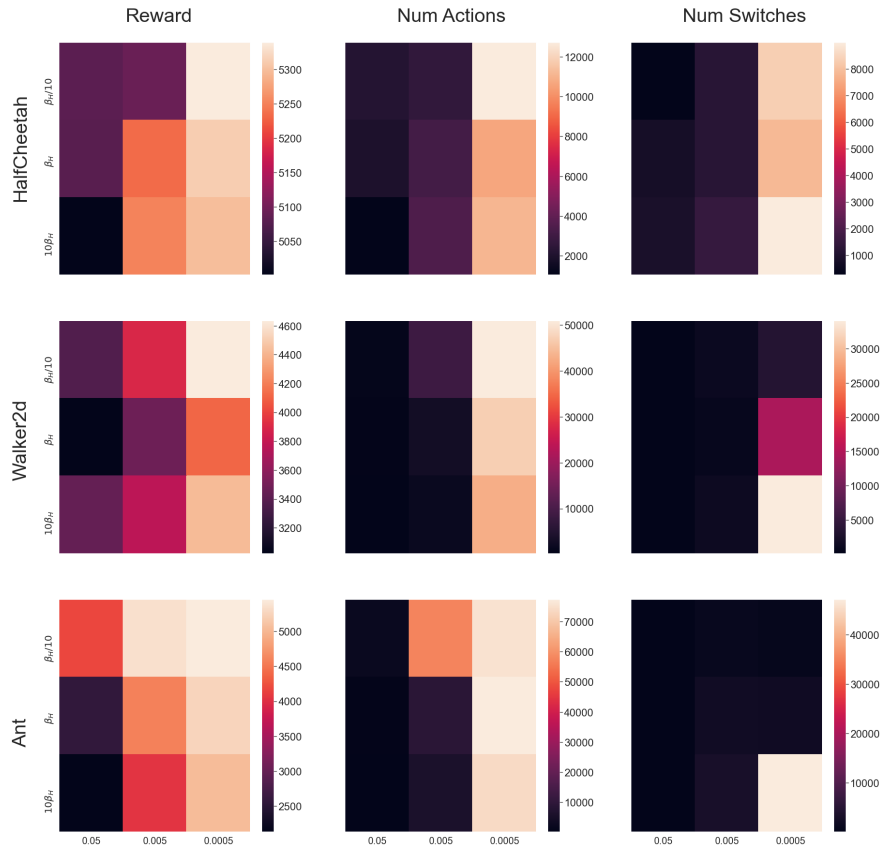


Figure A.1: LazyDAgger β_R and β_H sensitivity heatmaps across the 3 MuJoCo environments. The x-axis denotes β_H and the y-axis denotes β_R. Note that β_R is a function of β_H. Each of the 3 environments was run 9 times with the different settings of β_R and β_H. As in Figure 2.3 we plot test reward, number of supervisor actions, and number of context switches.

of point masses. Self-collision is implemented by applying a repulsive force between points that are sufficiently close together. Blender (`https://blender.org/`) is used to render the fabric in $100 \times 100 \times 3$ RGB image observations. See Figure 2.4 for an example observation. The actions are 4D vectors consisting of a pick point $(x, y) \in [-1, 1]^2$ and a place point $(\Delta x, \Delta y) \in [-1, 1]^2$, where $(x, y) = (-1, -1)$ corresponds to the bottom left corner of the plane while $(\Delta x, \Delta y)$ is multiplied by 2 to allow crossing the entire plane. In simulation, we initialize the fabric with coverage $41.1 \pm 3.4\%$ in the hardest (Tier 3) state distribution in [227] and end episodes if we exceed 10 time steps, cover at least 92% of the plane, are at least 20% out of bounds, or have exceeded a tearing threshold in one of the springs. We use the same algorithmic supervisor as [227], which repeatedly picks the coordinates of the corner furthest from its goal position and pulls toward this goal position. To facilitate transfer to the real world, we use the domain randomization techniques in [227] to vary the following parameters:

- Fabric RGB values uniformly between (0, 0, 128) and (115, 179, 255), centered around blue.
- Background plane RGB values uniformly between (102, 102, 102) and (153, 153, 153).
- RGB gamma correction uniformly between 0.7 and 1.3.
- Camera position (x, y, z) as $(0.5+\delta_1, 0.5+\delta_2, 1.45+\delta_3)$ meters, where each δ_i is sampled from $\mathcal{N}(0, 0.04)$.
- Camera rotation with Euler angles sampled from $\mathcal{N}(0, 90^\circ)$.
- Random noise at each pixel uniformly between -15 and 15.

For consistency, we use the same domain randomization in our sim-to-sim ("simulator to simulator") fabric smoothing experiments in Section 2.6.

Actor Policy and Discrepancy Classifier

The actor policy is a convolutional neural network with the same architecture as [227], i.e. four convolutional layers with 32 3x3 filters followed by four fully connected layers. The parameters, ignoring biases for simplicity, are:

```
policy/convnet/c1   864 params (3, 3, 3, 32)
policy/convnet/c2   9216 params (3, 3, 32, 32)
policy/convnet/c3   9216 params (3, 3, 32, 32)
policy/convnet/c4   9216 params (3, 3, 32, 32)
policy/fcnet/fc1    3276800 params (12800, 256)
policy/fcnet/fc2    65536 params (256, 256)
policy/fcnet/fc3    65536 params (256, 256)
policy/fcnet/fc4    1024 params (256, 4)
Total model parameters: 3.44 million
```

The discrepancy classifier reuses the actor's convolutional layers by taking a forward pass through them. We do not backpropagate gradients through these layers when training the classifier, but rather fix these parameters after training the actor policy. The rest of the classifier network has three fully connected layers with the following parameters:

```
policy/fcnet/fc1    3276800 params (12800, 256)
policy/fcnet/fc2    65536 params (256, 256)
policy/fcnet/fc3    1024 params (256, 4)
Total model parameters: 3.34 million
```

Training

Due to the large amount of data required to train fabric smoothing policies, we pretrain the *actor policy* (not the discrepancy classifier) in simulation. The learned policy is then fine-tuned to the new environment while the discrepancy classifier is trained from scratch. Since the algorithmic supervisor can be queried cheaply, we pretrain with DAgger as in [227]. To further accelerate training, we parallelize environment interaction across 20 CPUs, and before DAgger iterations we pretrain with 100 epochs of Behavior Cloning on the dataset of 20,232 state-action pairs available at [227]'s project website. Additional training hyperparameters are given in Table A.2 and the learning curve is given in Figure A.2.

Hyperparameter	Value
BC Epochs	100
DAgger Epochs	100
Parallel Environments	20
Gradient Steps per Epoch	240
Env Steps per Env per DAgger Epoch	20
Batch Size	128
Replay Buffer Size	5e4
Learning Rate	1e-4
L2 Regularization	1e-5

Table A.2: **DAgger Hyperparameters.** After Behavior Cloning, each epoch of DAgger (1) runs the current policy and collects expert labels for 20 time steps in each of 20 parallel environments and then (2) takes 240 gradient steps on minibatches of size 128 sampled from the replay buffer.

Experiments

In sim-to-sim experiments, the initial policy is trained on a 16x16 grid of fabric in a range of colors centered around blue with a spring constant of $k = 10,000$. We then adapt this policy to a new simulator with different physics parameters and an altered visual appearance. Specifically, in the new simulation environment, the fabric is a higher fidelity 25x25 grid with a lower spring constant of $k = 2,000$ and a color of (R, G, B) = (204, 51, 204) (i.e. pink), which is outside the range of colors produced by domain randomization (Section A.3). Hyperparameters are given in Table A.3.

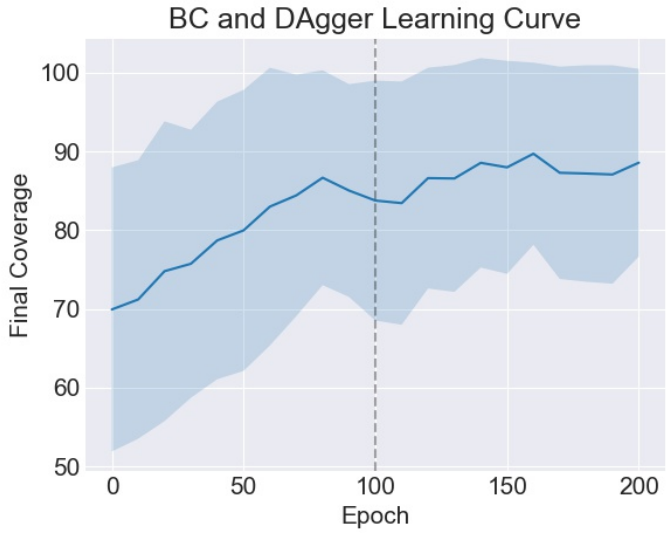


Figure A.2: Behavior Cloning and DAgger performance across 10 test episodes evaluated every 10 epochs. Shading indicates 1 standard deviation. The first 100 epochs (left half) are Behavior Cloning epochs and the second 100 (right half) are DAgger epochs.

Hyperparameter	Value
N	10
T	20
β_H	0.001
β_R	β_H
σ^2	0.05
Initial $\lvert\mathcal{D}\rvert$	1050
Initial $\lvert\mathcal{D}_{\mathrm{S}}\rvert$	450
Batch Size	50
Gradient Steps per Epoch	200
π Learning Rate	1e-4
f Learning Rate	1e-3
L2 Regularization	1e-5

Table A.3: Hyperparameters for sim-to-sim fabric smoothing experiments, where the first 5 rows are LazyDAgger hyperparameters in Algorithm 1. Initial dataset sizes and batch size are in terms of images *after* data augmentation, i.e. scaled up by a factor of 15 (see Section A.4). Note that the offline data is split 70%/30% as in Section A.1.

A.4 Fabric Manipulation with the ABB YuMi

Experimental Setup

We manipulate a brown 10" by 10" square piece of fabric with a single parallel jaw gripper as shown in Figure 2.1. The gripper is equipped with reverse tweezers for more precise picking of deformable materials. Neural network architecture is consistent with Section A.3 for both actor and safety classifier. We correct pick points that nearly miss the fabric by mapping to the nearest point on the mask of the fabric, which we segment from the images by color. To convert neural network actions to robot grasps, we run a standard camera calibration procedure and perform top-down grasps at a fixed depth. By controlling the width of the tweezers via the applied force on the gripper, we can reliably pick only the top layer of the fabric at a given pick point. We provide LazyDAgger-Execution hyperparameters in Table A.4.

Image Processing Pipeline

In the simulator, the fabric is smoothed against a light background plane with the same size as the fully smoothed fabric (see Figure 2.4). Since the physical workspace is far larger than the fabric, we process each RGB image of the workspace by (1) taking a square crop, (2) rescaling to 100×100, and (3) denoising the image. Essentially we define a square crop of the workspace as the region to smooth and align against, and assume that the fabric starts in this region. These processed images are the observations that fill the replay buffer and are passed to the neural networks.

User Interface

When the system solicits human intervention, an interactive user interface displays a scaled-up version of the current observation. The human is able to click and drag on the image to provide a pick point and pull vector, respectively. The interface captures the input as pixel locations and analytically converts it to the action space of the environment (i.e. $a \in [-1, 1]^4$) for the robot to execute. See Figure A.3 for a screen capture of the user interface.

Data Augmentation

To prevent overfitting to the small amount of real data, before adding each state-action pair to the replay buffer, we make 10 copies of it with the following data augmentation procedure, with transformations applied in a random order:

- Change contrast to 85-115% of the original value.
- Change brightness to 90-110% of the original value.

Figure A.3: The user interface for human interventions. The current observation of the fabric state from the robot's perspective is displayed, with an overlaid green arrow indicating the action the human has just specified.

- Change saturation to 95-105% of the original value.
- Add values uniformly between -10 and 10 to each channel of each pixel.
- Apply a Gaussian blur with σ between 0 and 0.6.
- Add Gaussian noise with σ between 0 and 3.
- With probability 0.8, apply an affine transform that (1) scales each axis independently to 98-102% of its original size, (2) translates each axis independently by a value between -2% and 2%, and (3) rotates by a value between -5 and 5 degrees.

Hyperparameter	Value
β_H	0.004
β_R	β_H
$\lvert\mathcal{D}\rvert$	875
$\lvert\mathcal{D}_{\mathrm{S}}\rvert$	375
Batch Size	50
Gradient Steps per Epoch	125
π Learning Rate	1e-4
f Learning Rate	1e-3
L2 Regularization	1e-5

Table A.4: Hyperparameters for physical fabric experiments provided in the same format as Table A.3. Since this is at execution time, N, T and σ^2 hyperparameters do not apply.

Appendix B

Appendix for Chapter 3

In Appendix B.1, we discuss algorithmic details for ThriftyDAgger and all comparisons. Then, Appendix B.2 discusses implementation and hyperparameter details for all algorithms. In Appendix B.3, we provide additional details about the simulation and physical experiment domains, and in Appendix B.4, we describe the protocol and detailed results from the conducted user study.

B.1 Algorithm Details

Here we provide a detailed algorithmic description of ThriftyDAgger and all comparisons.

ThriftyDAgger

The full pseudocode for ThriftyDAgger is provided in Algorithm 3. ThriftyDAgger first initializes π_r via Behavior Cloning on offline transitions ($\mathcal{D}_h$ from the human supervisor, π_h) (line 1-2). Then, π_r collects an initial offline dataset $\mathcal{D}_r$ from the resulting π_r, initializes $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ by optimizing Equation (3.5) on $\mathcal{D}_r \cup \mathcal{D}_h$, and initializes parameters $\beta_H, \beta_R, \delta_h$, and δ_r as in Section 3.4 (lines 3-5). We then collect data for N episodes, each with up to T timesteps. In each timestep of each episode, we determine whether robot policy π_r or human supervisor π_h should be in control using the procedure in Section 3.4 (lines 10-20). Transitions in autonomous mode are aggregated into $\mathcal{D}_r$ while transitions in supervisor mode are aggregated into $\mathcal{D}_h$. Episodes are terminated either when the robot reaches a valid goal state or has exhausted the time horizon T. At this point, we re-initialize the policy to autonomous mode and update parameters $\beta_H, \beta_R, \delta_h$, and δ_r as in Section 3.4 (lines 21-23). After each episode, π_r is updated via supervised learning on $\mathcal{D}_h$, while $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ is updated on $\mathcal{D}_r \cup \mathcal{D}_h$ to reflect the task success probability of the resulting π_r (lines 24-26).

Behavior Cloning

We train policy π_R via direct supervised learning with a mean-squared loss to predict reference control actions given a dataset of (state, action) tuples. Behavior Cloning is trained only on full expert demonstrations collected offline from π_H and is not allowed access to online interventions. Thus, Behavior Cloning is trained only on dataset $\mathcal{D}_h$ (line 1, Algorithm 3) and the policy is frozen thereafter. In our simulation experiments, Behavior Cloning is given 50% more offline data than the other algorithms for a more fair comparison, such that the amount of additional offline data is approximately equal to the average amount of online data provided to the other algorithms.

SafeDAgger

SafeDAgger [276] is an interactive imitation learning algorithm which selects between autonomous and supervisor mode using a classifier f that discriminates between "safe" states, for which π_R's proposed action is within some threshold β_H of that proposed by supervisor policy π_H, and "unsafe" states, for which this action discrepancy exceeds β_H. SafeDAgger learns this classifier using dataset $\mathcal{D}_h$ from Algorithm 3, and updates f online as $\mathcal{D}_h$ is expanded through human interventions. During policy rollouts, if f marks a state as safe, the robot policy is executed (autonomous mode), while if f marks a state as unsafe, the supervisor is queried for an action. While this approach can be effective in some domains [276], prior work [98] suggests that this intervention criterion can lead to excessive context switches between the robot and supervisor, and thus impose significant burden on a human supervisor. As in ThriftyDAgger and other DAgger [215] variants, SafeDAgger updates π_R on an aggregated dataset of all transitions collected by the supervisor (analogous to $\mathcal{D}_h$ in Algorithm 3).

LazyDAgger

LazyDAgger [98] builds on SafeDAgger [276] and trains the same action discrepancy classifier f to determine whether the robot and supervisor policies will significantly diverge at a given state. However, LazyDAgger introduces a few modifications to SafeDAgger which lead to lengthier and more informative interventions in practice. First, LazyDAgger observes that when the supervisor has control of the system (supervisor mode), querying f for estimated action discrepancy is no longer necessary since we can simply query the robot policy at any state during supervisor mode to obtain a true measure of the action discrepancy between the robot and supervisor policies. This prevents exploiting approximation errors in f when the supervisor is in control. Second, LazyDAgger introduces an asymmetric switching condition between autonomous and supervisor control, where switches are executed from autonomous to supervisor mode if f indicates that the predicted action discrepancy is above β_H, but switches are only executed from supervisor mode back to autonomous mode if the true action discrepancy is below some value $\beta_R < \beta_H$. This encourages lengthier interven-

Algorithm 3 ThriftyDAgger

Require: Number of episodes N, time horizon T, supervisor policy π_H, desired context switching rate α_h

Collect offline dataset $\mathcal{D}_h$ of (s, a^h) tuples with π_H
Initialize π_R via Behavior Cloning on $\mathcal{D}_h$
Collect offline dataset $\mathcal{D}_r$ of (s, a^r) tuples with π_R
Initialize $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ by optimizing Equation (3.4) on $\mathcal{D}_r \cup \mathcal{D}_h$
Optimize $\beta_H, \beta_R, \delta_h, \delta_r$ on $\mathcal{D}_h$ ▷ Automatic tuning based on α_h (Section 3.4)
for $i \in \{1, \dots N\}$ **do**
    Initialize s_0, Mode $\leftarrow$ Autonomous
    for $t \in \{1, \dots T\}$ **do**
        $a^r_t = \pi_R(s_t)$
        if Mode = Supervisor or Intervene(s_t, δ_h, β_H) **then** ▷ Determine control mode (Section 3.4)
            $a^h_t = \pi_H(s_t)$
            $\mathcal{D}_h \leftarrow \mathcal{D}_h \cup \{(s_t, a^h_t)\}$
            Execute a^h_t
            if Cede(s_t, δ_r, β_R) **then** ▷ Default control mode for next timestep (Section 3.4)
                Mode $\leftarrow$ Autonomous
            else
                Mode $\leftarrow$ Supervisor
        else
            Execute a^r_t
            $\mathcal{D}_r \leftarrow \mathcal{D}_r \cup \{(s_t, a_t)\}$
        if Terminal state reached **then**
            Exit Loop, Mode $\leftarrow$ Autonomous
            Recompute $\beta_H, \beta_R, \delta_h$ ▷ Automatic tuning based on α_h (Section 3.4)
    $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t, a^h_t) \sim \mathcal{D}_h} [\mathcal{L}(\pi_R(s_t), \pi_H(s_t))]$
    Collect $\mathcal{D}_r$ offline with robot policy π_R
    Update $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ on $\mathcal{D}_r \cup \mathcal{D}_h$ ▷ Update Q-function via Equation (3.6)

tions, leading to fewer context switches between autonomous and supervisor modes. Finally, LazyDAgger injects noise into supervisor actions in order to spread the distribution of states in which reference controls from the supervisor are available. ThriftyDAgger builds on the asymmetric switching criterion introduced by LazyDAgger, but introduces a new switching criterion based on the estimated task success probability, which we found significantly improved performance in practice.

HG-DAgger

Unlike SafeDAgger, LazyDAgger, and ThriftyDAgger, which are robot-gated and autonomously determine when to solicit intervention requests, HG-DAgger is human-gated, and thus requires that the supervisor determine the timing and length of interventions. As in ThriftyDAgger, HG-DAgger updates π_R on an aggregated dataset of all transitions collected by the supervisor (analogous to $\mathcal{D}_h$ in Algorithm 3).

B.2 Hyperparameter and Implementation Details

Here we provide a detailed overview of all hyperparameter and implementation details for ThriftyDAgger and all comparisons to facilitate reproduction of all experiments. We also include code in the supplement, and will release a full open-source codebase after anonymous review.

ThriftyDAgger

Peg Insertion (Simulation): We initially populate $\mathcal{D}_h$ with 2,687 offline transitions, which correspond to 30 task demonstrations collected by an expert human supervisor to initialize the robot policy π_R. We represent π_R with an ensemble of 5 neural networks, trained on bootstrapped samples of data from $\mathcal{D}_h$ in order to quantify uncertainty for novelty estimation. Each neural network is trained using the Adam Optimizer (learning rate $1e-3$) with 5 training epochs, 500 gradient steps in each training epoch, and a batch size of 100. All networks consist of 2 hidden layers, each with 256 hidden units with ReLU activations, and a Tanh output activation.

The Q-function used for risk-estimation, $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, is trained with a batch size of 50, and batches are balanced such that 10% of all sampled transitions contain a state in the goal set. We train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with the Adam Optimizer, with a learning rate of $1e-3$ and discount factor $\gamma = 0.9999$. In order to train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, we collect 10 test episodes from π_R every 2,000 environment steps. We represent $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with a 2 hidden layer neural net in which each hidden layer has 256 hidden units with ReLU activations and with a sigmoid output activation. The state and action are concatenated before they are fed into $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$.

Cable Routing (Physical): We initially populate $\mathcal{D}_h$ with 1,381 offline transitions, corresponding to 25 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_R. We again represent π_R with an ensemble of 5 neural networks, trained on bootstrapped samples of data from $\mathcal{D}_h$ in order to quantify uncertainty for novelty estimation. Each neural network is trained using the Adam Optimizer (learning rate $2.5e{-}4$) with 5 training epochs, 300 gradient steps per training epoch, and a batch size of 64. All networks consist of 5 convolutional layers (format: $(\text{in\_channels}, \text{out\_channels}, \text{kernel\_size}, \text{stride})$): [(3,24,5,2), (24,36,5,2), (36,48,5,2), (48,64,3,1), (64,64,3,1)] followed by 4 fully connected layers (format: $(\text{in\_units}, \text{out\_units})$): [(64,100), (100,50), (50,10), (10,2)]. Here we utilize ELU (exponential linear unit) activations with a Tanh output activation.

The Q-function used for risk-estimation, $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, is trained with a batch size of 64 as well, and batches are balanced such that 10% of all sampled transitions contain a state in the goal set. We train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with the Adam Optimizer with a learning rate of $2.5e-4$ and discount factor $\gamma = 0.9999$. In order to train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, we collect 5 test episodes from π_R every 500 environment steps. We represent $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with a neural network with the same 5 convolutional layers as the policy networks above, but with the fully connected layers as follows (format:

(in_units, out_units)): $[(64 + 2, 100), (100, 50), (50, 10), (10, 1)]$. We concatenate the action with the state embedding resulting from the 5 convolutional layers (hence the 64 + 2) and feed the resulting concatenated embedding into the 4 fully connected layers above. We utilize ELU (exponential linear unit) activations with a sigmoid output activation.

Behavior Cloning

Peg Insertion (Simulation): For Behavior Cloning, we initially populate $\mathcal{D}_h$ with 4,004 offline transitions, corresponding to 45 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_R (note that this is more transitions than are provided to ThriftyDAgger). All other details are the same as ThriftyDAgger for training π_R.

Cable Routing (Physical): We train π_R with the same architecture and procedure as for ThriftyDAgger, but only on the initial offline data.

SafeDAgger

We use the same hyperparameters and architecture for training π_R as for ThriftyDAgger. Unlike ThriftyDAgger, SafeDAgger does not have a mechanism to automatically tune hyperparameters when provided an intervention budget α_h. Thus, we must specify a value for the switching threshold β_H. We use $\beta_H = 0.008$, since this is recommended in [276] as the value which was found to work well in experiments (in practice, this value marks about 20% of states as "unsafe").

LazyDAgger

We use the same hyperparameters and architecture for training π_R as for ThriftyDAgger. Unlike ThriftyDAgger, LazyDAgger does not have a mechanism to automatically tune hyperparameters when provided an intervention budget α_h. Thus, we must specify a value for both switching thresholds β_H and β_R. We use $\beta_H = 0.015$, $\beta_R = 0.25\beta_H$ and use a noise covariance matrix of $0.02\mathcal{N}(0, I)$ when injecting noise into the supervisor actions.

HG-DAgger

All hyperparameters and architectures are identical to those used for Behavior Cloning. Note that for HG-DAgger, the supervisor determines the timing and length of interventions.

B.3 Environment Details

Peg Insertion in Simulation

We evaluate our algorithm and baselines in the Robosuite environment (`https://robosuite.ai`) [280], which builds on MuJoCo [251] to provide a standardized suite of benchmark tasks for robot learning. Specifically, we consider the "Nut Assembly" task, in which a robot must grab a ring from a random initial pose and place it over a cylinder at a fixed location. We consider the variant of the task that considers only 1 ring and 1 target, though the simulator allows 2 rings and 2 targets. The states are $s \in \mathbb{R}^{51}$ and actions $a \in \mathbb{R}^5$ (translation in the XY-plane, translation in the Z-axis, rotation around the Z-axis, and opening or closing the gripper). The simulated robot arm is a UR5e, and the controller reaches a commanded pose via operational space control with fixed impedance. To avoid bias due to variable teleoperation speeds, we abstract 10 timesteps in the simulator into 1 environment step, and in supervisor mode we pause the simulation until keyboard input is received. Each episode is terminated upon successful task completion or when 175 actions are executed. Interventions are collected through a keyboard interface.

Physical Cable Routing

We also evaluate our algorithm on a visuomotor cable routing task with a da Vinci Research Kit surgical robot. We take RGB images of the scene with a Zivid One Plus camera inclined at about 45 degrees to the vertical. These images are cropped into a square and downsampled to 64×64 before they are passed to the neural network policy. The cable state is initialized to approximately the same shape (see Figure 3.2) with the cable initialized in the robot's gripper. The workspace is approximately 10 cm $\times$ 10 cm, and each component of the robot action $(\Delta x, \Delta y)$ is at most 1 cm in magnitude. To avoid collision with the 4 obstacles, we implement a "logical obstacle" as 1-cm radius balls around the center of each obstacle. Actions that enter one of these regions are projected to the boundary of the circle. Each episode is terminated upon successful task completion or 100 actions executed. When collecting data from the human supervisor, we calculate 2D pose deltas at 1 second intervals. Interventions are collected through a 7DOF teleoperation interface that matches the pose of the robot arm.

B.4 User Study Details

Here we detail the protocol for conducting user studies with ThriftyDAgger and comparisons and discuss qualitative results based on participants' answers to survey questions measuring their mental load and levels of frustration when using each of the algorithms.

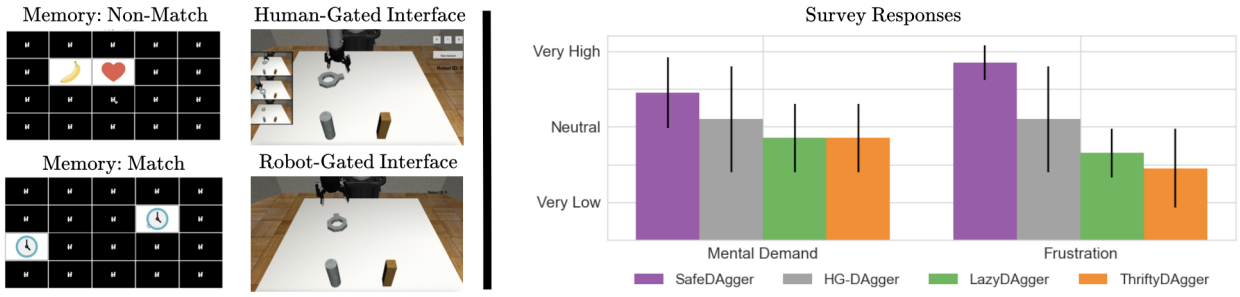


Figure B.1: **User Study Survey Results:** We illustrate the user study interface for the human-gated and robot-gated algorithms (left) and users' survey responses regarding their mental load and frustration (right) for each algorithm. Results suggest that users report similar levels of mental load and frustration for ThriftyDAgger and LazyDAgger, but significantly higher levels of both metrics for HG-DAgger and SafeDAgger. We hypothesize that the sparing and sustained interventions solicited by ThriftyDAgger and LazyDAgger lead to greater user satisfaction and comfort compared to algorithms which force the user to constantly monitor the system (HG-DAgger) or frequently context switch between teleoperation and the distractor task.

User Study Interface and NASA TLX Survey Results

Figure B.1 illustrates the interface used for the user study (left) and the survey results (right). The user study is performed with the same peg insertion environment used for simulation experiments, but with 3 robots performing the task in parallel. The base policy is initialized from 30 demos, as in the other simulation experiments. To speed up the task execution for the user study, each action has twice the magnitude as in the peg insertion experiments. Since this results in shorter trajectories that are easier for Behavior Cloning to accomplish, we also inject a small amount of Gaussian noise (covariance matrix $= 0.02\mathcal{N}(0, I)$).

In the human-gated study with HG-DAgger, participants are shown videos of all 3 robots attempting to perform the task in a side pane (Figure B.1, top right of left pane) and are instructed to monitor all of the robots and intervene when they deem it appropriate. In all robot-gated studies, participants are instructed to play the Concentration game until they hear a chime, at which point they are instructed to switch screens to the teleoperation interface. The Concentration game (also called Memory) is illustrated on the left of the left pane in Figure B.1: the objective is to find pairs of cards (all of which are initially face-down) which have matching pictures on their front side. Examples of a non-matching pair and a matching pair are illustrated in Figure B.1.

All robots which require interventions are placed in a FIFO queue, with participants serving intervention requests sequentially until no robot requires intervention. Thus, the participant may be required to provide interventions for multiple robots in succession if multiple robots are currently in the queue. When no robot requires assistance, the teleop-

eration interface turns black and reports that no robot currently needs help, at which point participants are instructed to return to the Concentration game. After each participant is subjected to all 4 conditions (SafeDAgger, LazyDAgger, ThriftyDAgger, and HG-DAgger) in a randomized order, we give each participant a NASA TLX survey asking them to rate their mental demand and frustration for each of the conditions on a scale of 1 (very low) - 5 (very high). Results (Figure B.1 right pane) suggest that ThriftyDAgger and LazyDAgger impose less mental demand and make participants feel less frustrated than HG-DAgger and SafeDAgger. During experiments, we found that participants found it cumbersome to keep track of all of the robots simultaneously in HG-DAgger, while the frequent context switches in SafeDAgger made participants frustrated since they were often unable to make much progress in the Concentration Game and felt that the robot repeatedly asked for interventions in very similar states.

Detailed Protocol

For the user study, we recruited 10 participants from the Berkeley graduate student and post-doc community aged 18-37, including members without any knowledge or experience in robotics or AI. All participants are first assigned a randomly selected user ID. Then, participants are instructed to play a 12-card game of Concentration (also known as Memory) (`https://www.helpfulgames.com/subjects/brain-training/memory.html`) in order to learn how to play. Then, users are given practice with both the robot-gated and human-gated teleoperation interfaces. To do this, the operator of the study (one of the authors) performs one episode of the task in the robot-gated interface and briefly explains how to control the human-gated interface. Then, participants are instructed to perform one practice episode in the robot-gated teleoperation interface and spend a few minutes exploring the human-gated interface until they are confident in the usage of both interfaces and in how to teleoperate the robots. In the robot-gated experiments, participants are instructed to play Concentration when no robot asks for help, but to immediately switch to helping the robot whenever a robot asks for help. In the human-gated experiment with HG-DAgger, participants are instructed to continuously monitor all of the robots and perform interventions which they believe will maximize the number of successful episodes. During the robot-gated study, participants play the 24-card version of Concentration between robot interventions. If a participant completes the game, new games of Concentration are created until a time budget of robot interactions is hit. Then for each condition, the participant is scored based on (1) the number of times the robot successfully completed the task and (2) the number of total matching pairs the participant found across all games of Concentration.

Appendix C

Appendix for Chapter 5

C.1 Mathematical Details of the IFL Problem Formulation

Recall that ROHE takes the expectation over a distribution of trajectories, $p_{\omega,\theta_0}(\tau)$, where each trajectory $\tau = (\mathbf{s}^0, \mathbf{a}^0, ..., \mathbf{s}^T, \mathbf{a}^T)$ is composed of consecutive task episodes separated by resets and where the state-action tuples come from both π_θ and π_H. This distribution of trajectories is induced by ω and θ_0 because θ_0 parameterizes the initial robot policy π_{θ_0}, and ω affects the states that comprise D_H^t, which updates the robot policy π_θ for subsequent timesteps. In this section, we derive the mathematical relationship between the trajectory distribution $\tau \sim p_{\omega,\theta_0}(\tau)$ and the allocation policy ω.

Given an allocation policy ω, the human policy π_H, and the robot policy π_{θ_t} at each timestep t, the joint hybrid human-robot policy of all robots can be expressed as

$$\pi_{H\cup R}^t(\mathbf{s}) = \begin{bmatrix} \pi_{\theta_t}(s_1)(1 - \mathbb{1}_{\omega(\mathbf{s}^t,\pi_{\theta_t},\boldsymbol{\alpha}^{t-1},\mathbf{x}^t)_1}) + \pi_H(s_1)\mathbb{1}_{\omega(\mathbf{s}^t,\pi_{\theta_t},\boldsymbol{\alpha}^{t-1},\mathbf{x}^t)_1} \\ \vdots \\ \pi_{\theta_t}(s_N)(1 - \mathbb{1}_{\omega(\mathbf{s}^t,\pi_{\theta_t},\boldsymbol{\alpha}^{t-1},\mathbf{x}^t)_N}) + \pi_H(s_N)\mathbb{1}_{\omega(\mathbf{s}^t,\pi_{\theta_t},\boldsymbol{\alpha}^{t-1},\mathbf{x}^t)_N} \end{bmatrix}, \tag{C.1}$$

where $\mathbb{1}_{(.)}$ is an indicator function that selects the robot policy π_{θ_t} if robot i is allocated to a human and selects the human policy $\pi_H(s)$ otherwise. For notational convenience, $\omega(\mathbf{s}^t, \pi_{\theta_t}, \boldsymbol{\alpha}^{t-1}, \mathbf{x}^t)_i := \sum_{j=1}^M \boldsymbol{\alpha}_{ij}^t \in \{0, 1\}$.

The trajectory distribution $p_{\omega,\theta_0}(\tau)$ can then be expressed as

$$p_{\omega,\theta_0}(\tau) = p_{\omega,\theta_0}(\mathbf{s}^0, \mathbf{a}^0, ..., \mathbf{s}^T, \mathbf{a}^T) \tag{C.2}$$

$$= \bar{p}^0(\mathbf{s}^0) \prod_{t=0}^{T} \pi_{H\cup R}^t(\mathbf{a}^t|\mathbf{s}^t) \prod_{t=0}^{T-1} \bar{p}(\mathbf{s}^{t+1}|\mathbf{s}^t, \mathbf{a}^t). \tag{C.3}$$

We comment that the soft and hard reset can be easily incorporated into the transition dynamics $\bar{p}$ depending on the task. For example, for constraint violations (i.e., hard resets)

Algorithm 4 Fleet-DAgger

Require: MDP $\mathcal{M}$, Number of robots N, Number of humans M, Priority function $\hat{p}$, Minimum teleoperation time t_T, Hard reset time t_R
Ensure: Allocation policy ω

function $\omega(\mathbf{s}^t, \pi_{\theta_t}, \boldsymbol{\alpha}^{t-1}, \mathbf{x}^t)$ ▷ The allocation policy ω returns a matrix $\boldsymbol{\alpha}^t \in \{0,1\}^{N \times M}$
  Compute priority scores of each robot: $\hat{p}(s_i^t, \pi_{\theta_t}) \quad \forall i = 1, ..., N$
  Initialize $\boldsymbol{\alpha}_{ij}^t = 0 \quad \forall i, j$
  for $i \in \{1, \ldots, N\}$ **do**
    for $j \in \{1, \ldots, M\}$ **do**
      if $\boldsymbol{\alpha}_{ij}^{t-1} = 1$ **then** ▷ For robots that were receiving assistance during the last timestep, check whether the minimum intervention time has lapsed using auxiliary information $\mathbf{x}^t$
        if Intervention type for robot i = `Hard reset` and Intervention duration $< t_R$ **then**
          $\boldsymbol{\alpha}_{ij}^t = 1$
        if Intervention type for robot i = `Teleop` and Intervention duration $< t_T$ **then**
          $\boldsymbol{\alpha}_{ij}^t = 1$
  Let $I = \{i : \sum_{j=1}^{M} \boldsymbol{\alpha}_{ij}^t = 1\}$ ▷ Set of robots that will continue with past assistance
  Let $J = \{j : \sum_{i=1}^{N} \boldsymbol{\alpha}_{ij}^t = 1\}$ ▷ Set of humans that will continue with past assistance
  Sort robot indices with positive priority scores that are not in I from highest to lowest, denoted as $\{i_1, i_2, ...\}$
  Let $k = 1$
  for $j \in \{1, \ldots, M\} \setminus J$ **do**
    $\boldsymbol{\alpha}_{i_k, j}^t = 1$
    $k = k + 1$
  return $\boldsymbol{\alpha}$

return ω

$c(s_i^t) = 1$, we can set $p(s_i^{t+k}|s_i^t, a_i^t) = \delta(s_i^t)$ for $1 \leq k \leq t_R - 1$ and $p(s_i^{t+T_R}|s_i^t, a_i^t) = p_i^0(s^0)$ where $\delta(\cdot)$ is the Dirac delta function and t_R is the hard reset time. Similarly, for goal-conditioned tasks with goal g, soft resets after achieving the goal can be expressed through the transition dynamics $p(s_i^{t+1}|s_i^t, a_i^t) = p_i^0(s^0)$ if $s_i^t \subseteq g$. For MDPs with finite time horizon where the environment resets when the maximum time horizon is reached, we can augment the state with additional time information that keeps track of the timestep in each episode, and reset the state when it times out. In this case, the MDP transition dynamics will be time-dependent: $p_t(s_i^{t+1}|s_i^t, a_i^t)$.

C.2 Fleet-DAgger Algorithm Details

In this section, we provide a detailed algorithmic description of Fleet-DAgger.

Fleet-DAgger uses priority function $\hat{p}$ and t_T to define an allocation policy ω. Concretely, it can be interpreted as a function F where $F(\hat{p}) = \omega$, i.e., a "meta-algorithm" (algorithm that outputs another algorithm) akin to function composition. The pseudocode of Fleet-DAgger is provided in Algorithm 4.

C.3 Additional Experiment Details

Simulation Hyperparameters

Implementations of C.U.R. and baselines are available in the code supplement, where the scripts are configured to run with the same hyperparameters as the simulation experiments in the main text. Recall that $N = 100$ robots, $M = 10$ humans, minimum intervention time $t_T = 5$, hard reset time $t_R = 5$, and operation time $T = 10000$. For reference, additional parameters are in Table C.1, where $|S|$ is the dimensionality of the (continuous) state space, $|A|$ is the dimensionality of the (continuous) action space, $\hat{r}$ is the risk threshold below which robots are assigned zero risk, $\hat{u}$ is the uncertainty threshold below which robots are assigned zero uncertainty, and t_I is the length of the initial C.U.R. period during which constraint violation is not prioritized.

Environment	$\|S\|$	$\|A\|$	$\hat{r}$	$\hat{u}$	t_I
Humanoid	108	21	0.5	0.05	1000
Anymal	48	12	0.5	0.05	250
AllegroHand	88	16	0.5	0.15	2500

Table C.1: Simulation environment hyperparameters.

Training Critic Q-Functions

Some IFL algorithms require pretraining a safety critic (C.U.R.) or goal critic (Fleet-Thrifty DAgger) to assist in supervisor allocation. Here we provide details on how we train these critics in practice. Additional details about training these critics in practice are also available in Recovery RL [247] and ThriftyDAgger [97] for the safety critic and goal critic respectively.

To collect a dataset of constraint violations, we simply run Behavior Cloning for a fixed amount of timesteps. Intuitively, since the initial robot policy π_{θ_0} is not highly performant, the robot should expect to encounter constraint violations, and these violations will occur within the state distribution visited by the robot fleet in the initial stages of online training. One could also inject noise into the BC policy to induce more constraint violations, or explicitly solicit human demonstrations of constraint violations as noted in [247]. For Humanoid, Anymal, and AllegroHand, we collect a dataset of 19625 transitions with 376 constraint violations, 19938 transitions with 63 constraint violations, and 19954 transitions with 47 constraint violations, respectively. The safety critic is then trained via Q-learning for 3000 gradient steps where a constraint-violating transition can be interpreted as incurring sparse reward $r = 1$ and all other transitions have reward $r = 0$. To reduce class imbalance issues, transitions are sampled from the replay buffer such that constraint-violating samples constitute 25% of the minibatch, which was found to be useful in practice in [247].

To collect a dataset of successes to pretrain the goal critic, we instead run the expert policy π_H, which is more likely to reach the goal. For AllegroHand, we collect a dataset of 19994 transitions with 489 successes. We then pretrain the goal critic in the same manner as the safety critic. Both the safety and goal critic continue to update during online training with the additional constraint violation and success data encountered.

Physical Experiment Protocol

We execute 3 trials of each of 4 algorithms (Behavior Cloning, Random, Fleet-Ensemble DAgger, C.U.R.) on the fleet of 4 robot arms for 250 timesteps each, for a total of $3 \times 4 \times 4 \times 250 = 12000$ individual pushing actions. Human teleoperation and hard resets were performed by the authors, where teleoperation is collected through an OpenCV (`https://opencv.org/`) graphical user interface and hard resets are physical adjustments of the cube toward the center of the workspace. Each of the 2 ABB YuMi robots is connected via Ethernet to a Linux machine on its local area network; the driver program connects to each machine over the Internet via the Secure Shell Protocol (SSH) to send robot actions and receive camera observations. The 2 YuMis are in different physical locations 0.5 miles apart, and all 4 arms execute actions concurrently with multiprocessing. $10\times$ data augmentation is performed on the initial offline dataset of 375 state-action pairs as well as the online data collected during execution as follows:

- Linear contrast uniformly sampled between 85% and 115%
- Add values uniformly sampled between -10 and 10 to each pixel value per channel
- Gamma contrast uniformly sampled between 90% and 110%
- Gaussian blur with σ uniformly sampled between 0.0 and 0.3
- Saturation uniformly sampled between 95% and 105%
- Additive Gaussian noise with σ uniformly sampled between 0 and $\frac{1}{80} \times 255$

Evaluating Trained Simulation Policies

Here we execute all policies from Section 5.6 after the 10,000 timesteps of online training for an additional 10,000 timesteps *without additional human teleoperation* to evaluate the quality of the learned robot policies in isolation. As in Section 5.6, $N = 100$ robots, $M = 10$ humans (for hard resets only), $t_R = 5$, and $T = 10000$, where allocation is performed by C-prioritization. We also include the expert policy performance (all $N = 100$ robots teleoperated by the trained PPO supervisor for $T = 10000$ steps) in the table for reference.

Results are in Table C.2. We find that the policy learned via C.U.R. outperforms baselines and approaches expert-level performance for Humanoid and Anymal, but is second most performant and significantly below expert-level performance for AllegroHand, indicating

Environment	Algorithm	Successes (↑)	Hard Resets (↓)	Idle Time (↓)
Humanoid	BC	0.0 ± 0.0	11925.3 ± 118.8	62473.7 ± 869.1
	Random	746.3 ± 40.5	340.0 ± 59.2	1700.0 ± 296.1
	Fleet-ED	617.7 ± 66.3	570.3 ± 139.0	2851.7 ± 694.8
	C.U.R.	**771.0 ± 25.5**	**289.3 ± 21.1**	**1446.7 ± 105.3**
	Expert	894	115	575
Anymal	BC	32.7 ± 0.5	1134.3 ± 33.9	5669.7 ± 170.0
	Random	207.3 ± 27.2	232.3 ± 89.6	1162.3 ± 449.1
	Fleet-ED	**257.3 ± 1.2**	109.0 ± 16.9	545.0 ± 84.4
	C.U.R.	**257.0 ± 8.8**	**64.3 ± 8.6**	**321.7 ± 42.9**
	Expert	293	1	5
AllegroHand	BC	70.0 ± 5.0	**523.0 ± 9.4**	**2614.3 ± 46.4**
	Random	**7360.3 ± 231.0**	2954.3 ± 131.0	14767.7 ± 653.2
	Fleet-ED	3032.0 ± 191.2	1764.7 ± 225.4	8818.0 ± 1129.7
	Fleet-TD	4296.3 ± 161.7	1397.7 ± 112.6	6988.0 ± 562.9
	C.U.R.	6032.7 ± 236.2	2343.7 ± 82.5	11715.0 ± 411.9
	Expert	21609	1202	6013

Table C.2: Robot policy performance for each of the algorithms in Section 5.6. We report cumulative successes, hard resets, and idle time. We do not report return on human effort as teleoperation is not performed for this experiment.

$T = 10,000$ timesteps is insufficient for learning a highly performant robot policy in this challenging environment.

C.4 Hyperparameter Sensitivity and Ablation Studies

In this section, we run additional simulation experiments in the IFL benchmark to study (1) ablations of the components of the C.U.R. algorithm (Figure C.1), (2) sensitivity to the ratio of number of robots N to number of humans M (Figure C.2), (3) sensitivity to minimum intervention time t_T (Figure C.3), and (4) sensitivity to hard reset time t_R (Figure C.4). All runs are averaged over 3 random seeds, where shading indicates 1 standard deviation.

Ablations: We test C.U.R.(-i), the C.U.R. algorithm without the initial period during which constraint violation is not prioritized. We also test all subsets of the C.U.R. priority function without the initial period. For example, U. indicates only prioritizing by uncertainty, and C.R. indicates prioritizing by constraint violations followed by risk (no uncertainty). Results suggest that C.U.R. outperforms all ablations in all environments in terms of ROHE and cumulative successes and is competitive in terms of hard resets and idle time. However,

as in the main text, C.U.R. and C. incur more hard resets in AllegroHand than alternatives, as again, prioritizing constraint violations for a hard environment where learning has not converged may ironically enable more opportunities for hard resets. Interestingly, while C.U.R. outperforms ablations in ROHE in AllegroHand for large T, U-prioritization's ROHE is significantly higher for small values of T. We observe that since U. achieves very low cumulative successes in the same time period, U. must be requesting an extremely small amount of human time early in operation, resulting in erratic ratio calculations.

Number of Humans: While keeping N fixed to 100 robots, we run C.U.R. with default hyperparameters and vary M to be 1, 5, 10, 25, and 50 humans. In the Humanoid and Anymal environment, as expected, cumulative successes increases with the number of humans. The performance boost gets smaller as M increases: runs with 25 and 50 humans have very similar performance. Despite lower cumulative successes, $M = 10$ achieves the highest ROHE, suggesting a larger set of humans provides superfluous interventions. We also observe that with only 1 human, the number of hard resets and idle time is very large, as the human is constantly occupied with resetting constraint-violating robots, which fail at a faster rate than the human can reset them. Finally, in the AllegroHand environment, the number of humans when $M \geq 5$ does not make much of a visible difference, perhaps due to the relatively high number of cumulative successes.

Minimum Intervention Time: We run C.U.R. with default hyperparameters but vary t_T to be 1, 5, 20, 50, 100, and 500 timesteps. We observe that both decreasing t_T from 5 to 1 and increasing t_T to 20 and beyond have a negative impact on the ROHE due to ceding control prematurely (in the former case) and superfluous intervention length (in the latter). Hard resets are low and idle time is high for large t_T as the humans are occupied providing long teleoperation interventions. This also negatively affects throughput, as cumulative successes falls for very large t_T. Long interventions may also be less useful training data, as in the limit these interventions reduce to more offline data (i.e., labels for states encountered under the human policy rather than that of the robot).

Hard Reset Time: Finally, we run C.U.R. with default hyperparameters but vary t_R to be 1, 5, 20, 50, 100, and 500 timesteps. As expected, the ROHE decreases as t_R increases, as more human effort is required to achieve the same return. The other metrics follow similar intuitive trends: increasing t_R results in a decrease in cumulative successes, decrease in hard resets, and increase in idle time.

Other Parameters: We also found that the batch size (256 in our experiments) and number of gradient steps per experiment timestep (1 in our experiments) significantly impact the policy learning speed as well as computation time, and the effect varies with the size of the fleet (N) and set of human supervisors (M). This is because N and M (and ω) determine how much new data is available at each time t, the batch size and number of gradient steps determine how much data to update the policy with at each time t, and updating the policy with backpropagation is the computational bottleneck in the IFLB simulation.

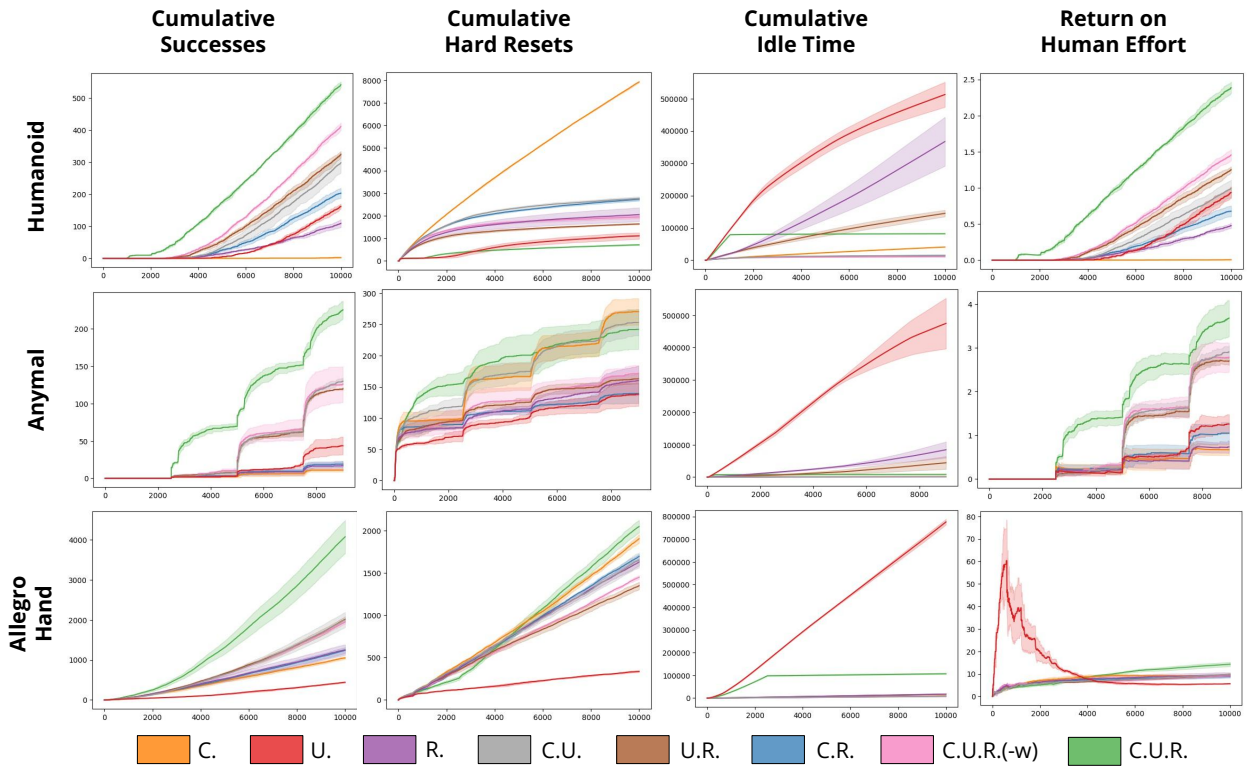


Figure C.1: **Ablations:** Simulation results in the Isaac Gym benchmark tasks with ablations of C.U.R., where the x-axis is timesteps from 0 to $T = 10,000$. We plot the metrics described in 5.6. The C.U.R. algorithm outperforms all ablations on all environments in terms of ROHE and cumulative successes (except AllegroHand ROHE for low T values) and is competitive with ablations for cumulative hard resets and idle time.

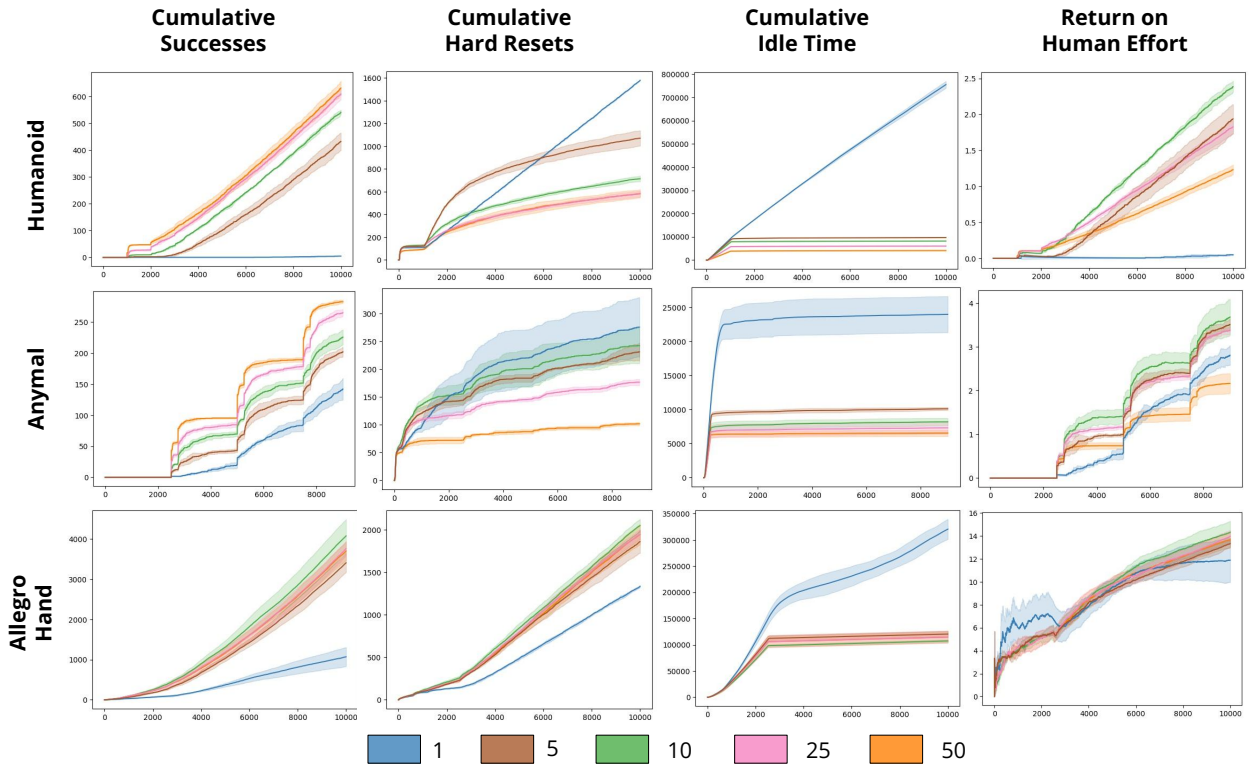


Figure C.2: **Number of Humans:** Simulation results in the Isaac Gym benchmark tasks with $N = 100$ robots and M human supervisors, where M varies and the x-axis is timesteps from 0 to $T = 10,000$.

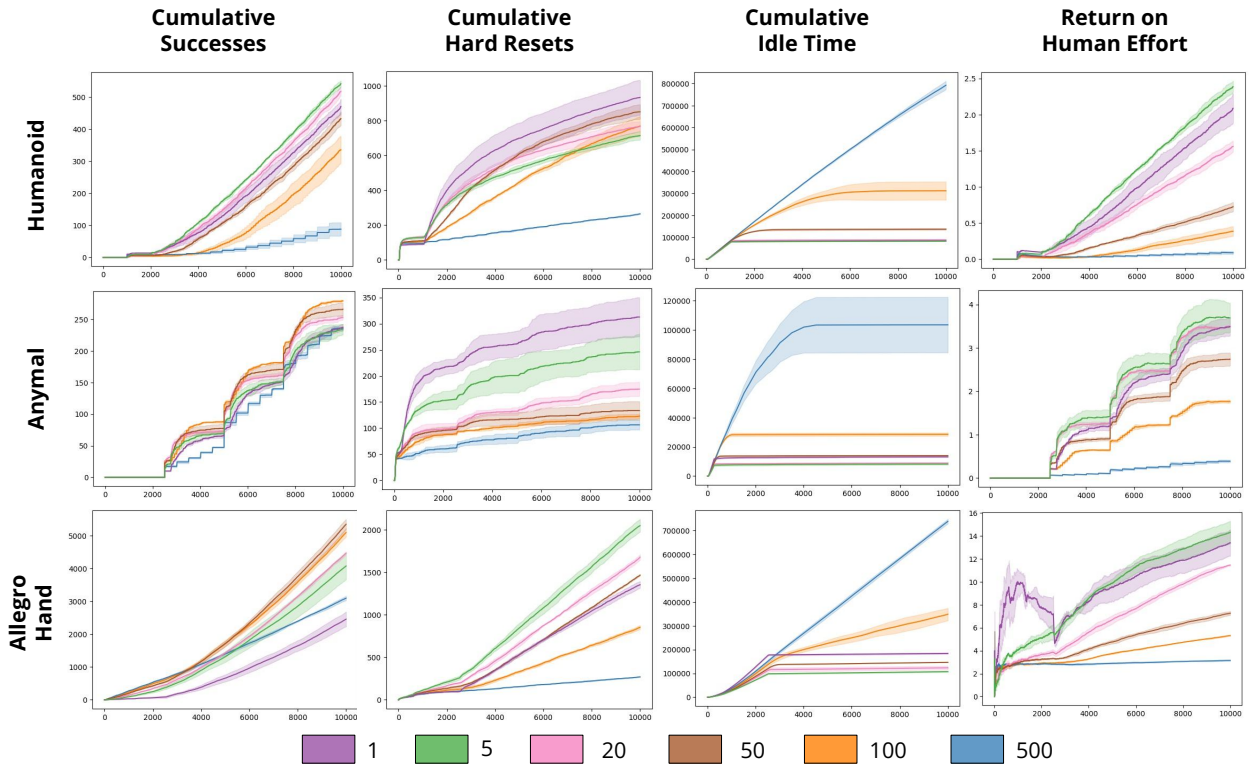


Figure C.3: **Minimum Intervention Time:** Simulation results in the Isaac Gym benchmark tasks for variations in minimum intervention time t_T, where the x-axis is timesteps from 0 to $T = 10,000$.

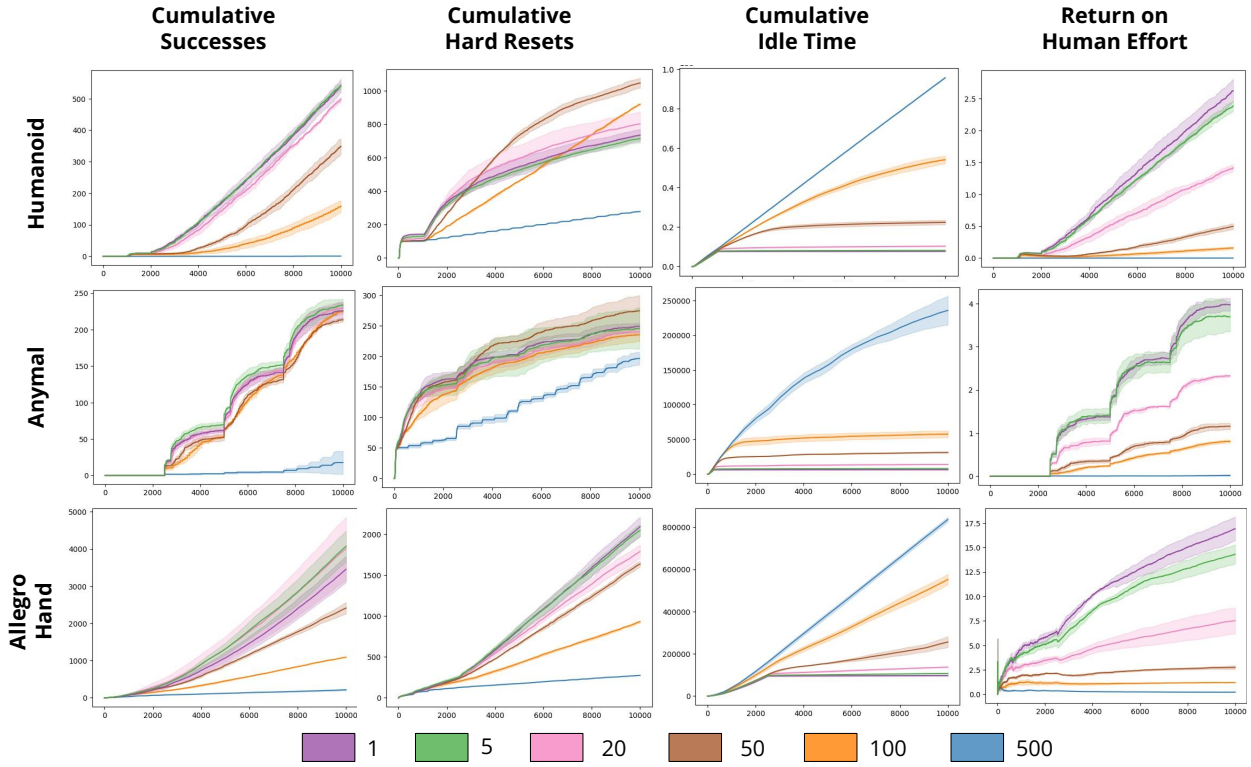


Figure C.4: **Hard Reset Time:** Simulation results in the Isaac Gym benchmark tasks for variations in hard reset time t_R, where the x-axis is timesteps from 0 to $T = 10,000$.

Appendix D

Appendix for Chapter 6

D.1 Jeffreys Divergence Identity

We derive the following identity from the main text:

Identity 1. *Let E_1 and E_2 be two energy-based models that respectively define distributions π_1 and π_2 according to Equation 6.4. Then,*

$$D_J\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) = \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right].$$

Proof. The proof follows from applying the definition of Jeffreys divergence to EBMs:

$$\begin{aligned}
D_J\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) &= D_{KL}\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) + D_{KL}\left(\pi_2(\cdot|s)\|\pi_1(\cdot|s)\right) \\
&= \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[\log\frac{\pi_1(a|s)}{\pi_2(a|s)}\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[\log\frac{\pi_2(a|s)}{\pi_1(a|s)}\right] \\
&= \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] - \log Z_1(s) + \log Z_2(s) \\
&\quad + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right] - \log Z_2(s) + \log Z_1(s) \\
&= \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right].
\end{aligned}$$

□

To provide more intuition on this identity, we plot the Jeffreys divergence for a pair of isotropic Gaussian energy functions in Figure D.1.

D.2 Additional Details on Implicit Models

Implicit BC trains an energy-based model E_θ on samples $\{s_i, a_i\}$ collected from the expert policies π_H. After generating a set of counter-examples $\{\tilde{a}_i^j\}$ for each s_i, Implicit BC minimizes the following InfoNCE [185] loss function:

$$\mathcal{L} = \sum_{i=1}^{N} -\log \hat{p}_\theta(a_i|s_i, \{\tilde{a}_i^j\}), \quad \hat{p}_\theta(a_i|s_i, \{\tilde{a}_i^j\}) := \frac{e^{-E_\theta(s_i,a_i)}}{e^{-E_\theta(s_i,a_i)} + \sum_j e^{-E_\theta(s_i,\tilde{a}_i^j)}}. \tag{D.1}$$

This loss is equivalent to the negative log likelihood of the training data, where the partition function $Z(s)$ is estimated with the counter-examples. Florence et al. [75] propose three techniques for generating these counter-examples $\{\tilde{a}_i^j\}$ and performing inference over the learned model E_θ; we choose gradient-based Langevin sampling [262] with an additional gradient penalty loss for training in this work as Florence et al. [75] demonstrate that it scales with action dimensionality better than the alternate methods. This is a Markov Chain Monte Carlo (MCMC) method with stochastic gradient Langevin dynamics. More details are available in Appendix B.3 of Florence et al. [75].

We use the following hyperparameters for implicit model training and inference:

Hyperparameter	Value
learning rate	0.0005
learning rate decay	0.99
learning rate decay steps	100
train counter examples	8
langevin iterations	100
langevin learning rate init.	0.1
langevin learning rate final	1e-5
langevin polynomial decay power	2
inference counter examples	512

Table D.1: Implicit model hyperparameters.

D.3 Uncertainty Estimation with Larger Ensembles

Prior works using ensembles of explicit models to estimate epistemic uncertainty [51, 171, 97] typically employ larger ensembles of $n \geq 5$ models, whereas IIFL uses $n = 2$. We wish to evaluate the impact of this smaller number of models. However, the Jeffreys divergence is only defined for two distributions, and while other divergence measures (e.g. Jensen-Shannon) can be generalized to an arbitrary number of distributions, they typically require knowledge of the intractable partition functions of the distributions. Accordingly, we consider estimating the uncertainty of $n = 5$ implicit models by computing the average of the Jeffreys divergences between every pairwise combination of models. Figure D.2 provides intuition on this measure, and we provide information on computation time in Section D.4.

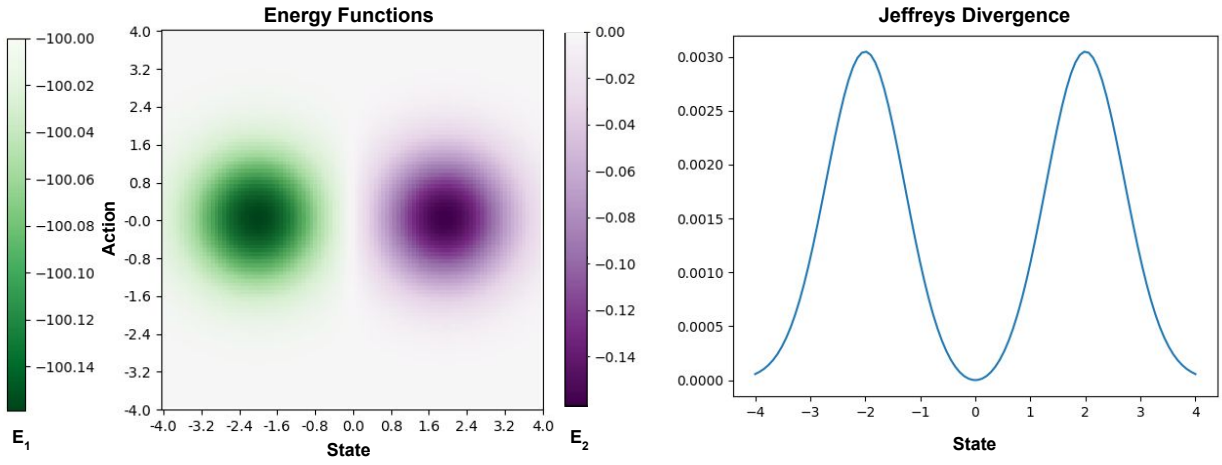
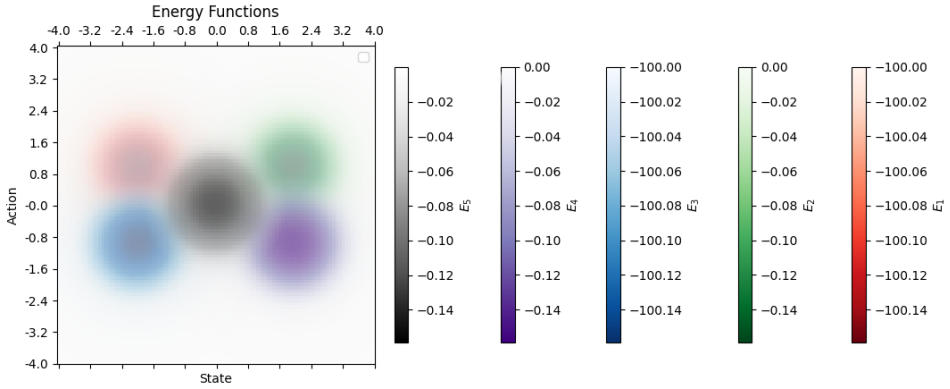
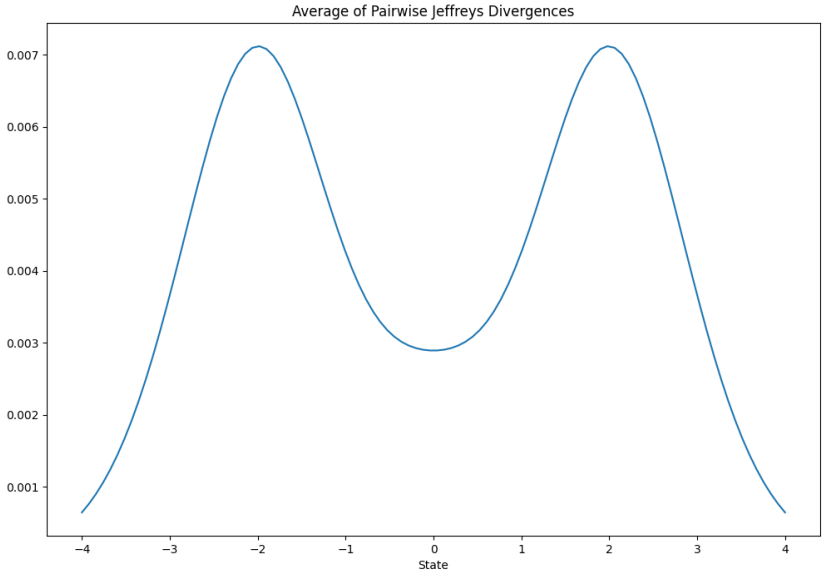


Figure D.1: Consider a pair of isotropic Gaussian energy functions $E_1(s, a)$ and $E_2(s, a)$ in green and purple respectively, where each function is a negated Gaussian probability density function and E_1 adds a uniform offset of $Z = -100$ to all values (Left). Using numerical integration to directly compute the expectations in the Jeffreys divergence identity (Identity 1), at each state we calculate the distance between the implicit policies defined by the two energy functions (Right). As intuition suggests, the divergence peaks at the mean of each Gaussian (where one energy function is highest and the other is near zero) and approaches zero where the energy functions are the same (at the center and edges of the state space). Note the symmetric structure of the Jeffreys curve, which produces identical values regardless of the offset Z.

We evaluate the effect of adding more models by comparing the estimate of the Jeffreys divergence with $n = 2$ models and the averaged estimate with $n = 5$ models to the L2 distance between the robot policy's proposed action and the expert policy's action at the same state. While ground truth epistemic uncertainty is intractable to calculate, the ground truth action discrepancy between the human and robot can provide a correlate of uncertainty: higher discrepancy corresponds to higher uncertainty. The results are shown in Figure D.3. We observe that both ensemble sizes are positively correlated with action discrepancy, and that the ensemble with $n = 5$ models has a higher correlation ($r = 0.804$) than the ensemble with $n = 2$ models ($r = 0.688$). We also observe that the $n = 5$ ensemble has lower variance than $n = 2$: the standard deviation is 0.176 compared to 0.220. These results suggest that larger ensembles can improve the uncertainty estimation at the cost of increased computation time (2.6× in Section D.4).



(a) Consider 5 isotropic Gaussian energy functions, each a negative Gaussian probability density function with some offset.



(b) We use numerical integration to calculate at each state the Jeffreys divergences between each of the $\binom{5}{2} = 10$ unique pairs of models, and report the average value. As intuition suggests, the calculated uncertainty is highest at states -2 and 2.

Figure D.2

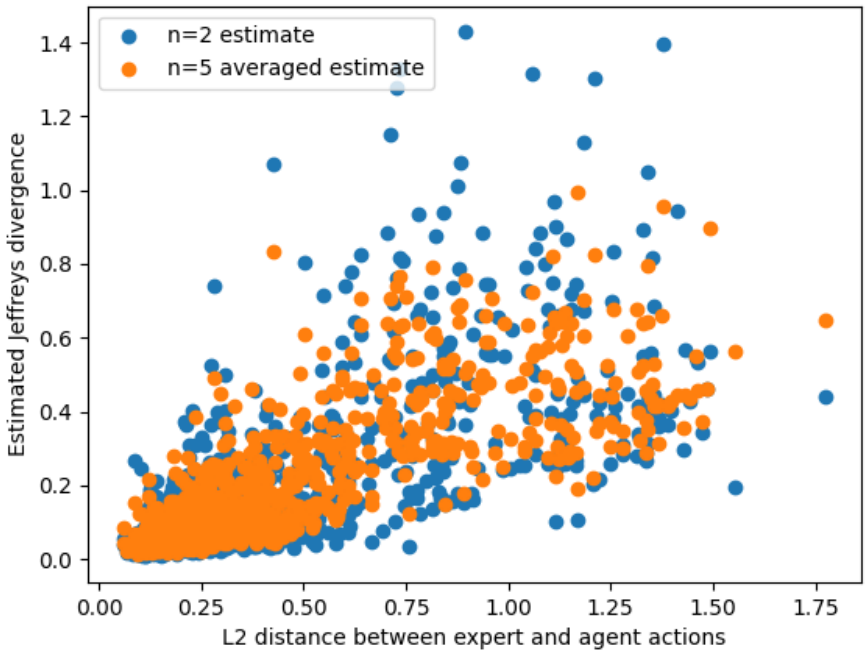


Figure D.3: We plot the Jeffreys divergence estimates and the ground truth action discrepancies at the first 1000 states visited by a robot with a unimodal policy. Both variants of the Jeffreys divergence calculation are positively correlated with the $L2$ distance between the robot policy's and expert policy's actions. In the $n = 2$ case, the correlation coefficient is $r = 0.688$; in the $n = 5$ case, the correlation coefficient is $r = 0.804$, indicating that additional models can make the ensemble more predictive of when the agent will deviate from the expert (at the cost of increased computation time).

D.4 Additional Experimental Details

IFL Benchmark Hyperparameters

Implementations of Implicit Interactive Fleet Learning and baselines are available in the code supplement and are configured to run with the same hyperparameters we used in the experiments. To compute the uncertainty thresholds $\hat{u}$ for Explicit IFL and IIFL (see Section 8.3.1 in [99] for definition), we run Explicit BC and Implicit BC respectively with $N = 100$ robots for $T = 1000$ timesteps and choose the 99th percentile value among all 100×1000 uncertainty values. The FrankaCubeStack environment sets these thresholds to zero since there are no constraint violations (i.e., this sorts robot priority by uncertainty alone). See Table D.2 for these values, state and action space dimensionality, and other hyperparameters. The batch size is 512 and all algorithms pretrain the policy for $N/2$ gradient steps, where N

is the number of data points in the 10 offline task demonstrations. Finally, as in prior work [99], the Random IIFL baseline is given a human action budget that approximately equals the average amount of human supervision solicited by IIFL. See the code for more details.

Environment	$\|S\|$	$\|A\|$	Explicit $\hat{u}$	Implicit $\hat{u}$
BallBalance	24	3	0.1179	0.1206
Ant	60	8	0.0304	0.9062
Anymal	48	12	0.0703	2.2845
FrankaCubeStack	19	7	0.0	0.0

Table D.2: Simulation environment hyperparameters.

FrankaCubeStack Environment

The scripted supervisor for FrankaCubeStack is defined in `human_action()` of `env/isaacgym/franka_cube_stack.py` in the code supplement. Using known pose information and Cartesian space control, the supervisor policy does the following, where Cube A is to be stacked on Cube B: (1) move the end effector to a position above Cube A; (2) rotate into a pre-grasp pose; (3) descend to Cube A; (4) lift Cube A; (5) translate to a position above Cube B; (6) place Cube A on Cube B; and (7) release the gripper. Heterogeneity is concentrated in Step 2: while one supervisor rotates to an angle $\theta \in [0, \frac{\pi}{2}]$ that corresponds to a pair of antipodal faces of the cube, the others rotate to $\theta - \pi$, $\theta - \frac{\pi}{2}$, and $\theta + \frac{\pi}{2}$. See Figure D.4 for intuition. We also include results for only 2 hetereogeneous policies (θ and $\theta - \frac{\pi}{2}$) in Table D.3; results (in conjunction with Table 6.1) suggest that relative performance of IIFL over baselines remains approximately consistent as the number of modes varies and can improve as multimodality increases.

Algorithm	Avg. Reward	Task Successes	ROHE
BC	23.45 ± 0.99	0.0 ± 0.0	N/A
IBC	30.32 ± 2.78	0.0 ± 0.0	N/A
IFL	307.87 ± 118.59	9.3 ± 4.7	3.08 ± 1.19
IIFL-R	244.98 ± 32.58	0.0 ± 0.0	2.45 ± 0.33
IIFL	$\mathbf{604.17 \pm 263.06}$	$\mathbf{17.7 \pm 11.1}$	$\mathbf{6.04 \pm 2.63}$

Table D.3: Execution results from the FrankaCubeStack Isaac Gym environment with 2 heterogeneous supervisor policies (rather than 4).

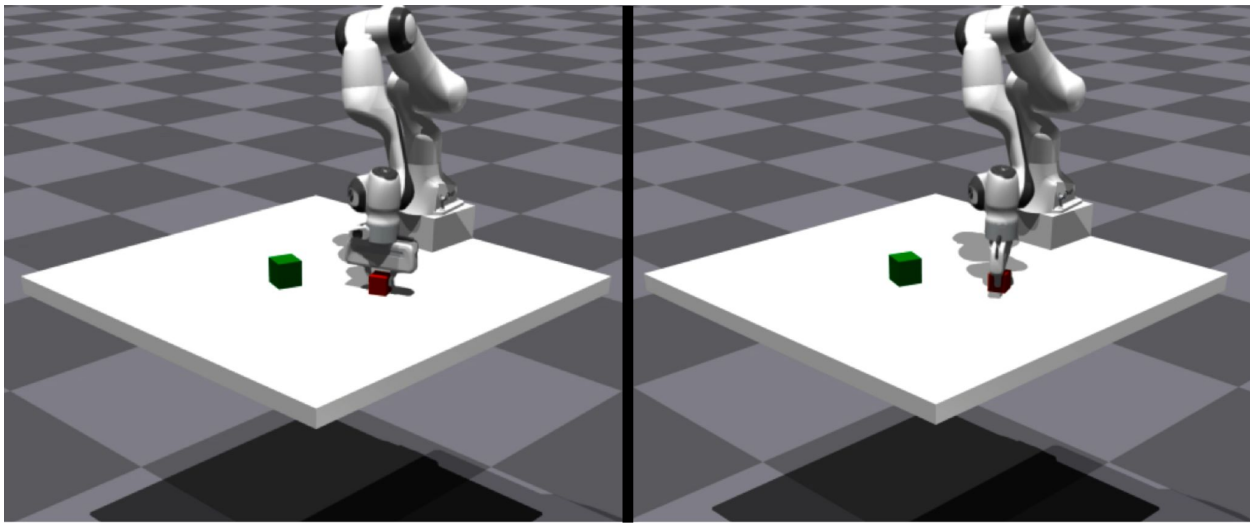


Figure D.4: The scripted heterogeneous supervisors for the FrankaCubeStack Isaac Gym environment pick different faces of the cube for the same cube pose.

Physical Experiment Protocol

We largely follow the physical experiment protocol in Hoque et al. [99] but introduce some modifications to human supervision. We execute 3 trials of each of 4 algorithms (Explicit BC, Implicit BC, Explicit IFL, Implicit IFL) on the fleet of 4 robot arms. Each trial lasts 150 timesteps (synchronous across the fleet) for a total of $3 \times 4 \times 4 \times 150 = 7200$ individual pushing actions. The authors provide human teleoperation and hard resets, which differ from prior work due to the continuous action space and the square obstacle in the center of the workspace. Teleoperation is done using an OpenCV (https://opencv.org/) GUI by clicking on the desired end point of the end-effector in the overhead camera view. Hard resets are physical adjustments of the cube to a randomly chosen side of the obstacle. IIFL is trained online with updated data at $t = 50$ and $t = 100$ while IFL is updated at every timestep (with an equivalent total amount of gradient steps) to follow prior work [99].

The rest of the experiment protocol matches Hoque et al. [99]. The 2 ABB YuMi robots are located about 1 km apart; a driver program uses the Secure Shell Protocol (SSH) to connect to a machine that is connected to the robot via Ethernet, sending actions and receiving camera observations. Pushing actions are executed concurrently by all 4 arms using multiprocessing. We set minimum intervention time $t_T = 3$ and hard reset time $t_R = 5$. All policies are initialized with an offline dataset of 3360 image-action pairs (336 samples collected by the authors with 10× data augmentation). 10× data augmentation on the initial offline dataset as well as the online data collected during execution applies the following transformations:

- Linear contrast uniformly sampled between 85% and 115%

- Add values uniformly sampled between -10 and 10 to each pixel value per channel
- Gamma contrast uniformly sampled between 90% and 110%
- Gaussian blur with σ uniformly sampled between 0.0 and 0.3
- Saturation uniformly sampled between 95% and 105%
- Additive Gaussian noise with σ uniformly sampled between 0 and $\frac{1}{80} \times 255$ 80×255

Computation Time

In Table D.4 we report the mean and standard deviation of various computation time metrics. All timing experiments were performed with $N = 100$ robots and averaged across $T = 100$ timesteps in the Ant environment on a single NVIDIA Tesla V100 GPU with 32 GB RAM. Training time is reported for a single gradient step with a batch size of 512. Note that with default hyperparameters, IFL trains an ensemble of 5 (explicit) models and IIFL trains an ensemble of 2 (implicit) models; hence, we also report the training time per individual model. IFL inference consists of a single forward pass through each of the 5 models, while IIFL inference performs 100 Langevin iterations; both of these are vectorized across all 100 robots at once. IFL uncertainty estimation also consists of a single forward pass through each of the 5 models while IIFL performs both Langevin iterations and 2 forward passes through each of the 2 models. While IIFL can provide policy performance benefits over IFL, we observe that it comes with a tradeoff of computation time, which may be mitigated with parallelization across additional GPUs. Furthermore, while uncertainty estimation is the bottleneck in IIFL, it is performed with sub-second latency for the entire fleet. This is significantly faster than alternatives such as directly estimating the partition function, which is both less accurate and slower; we measure it to take an average of 7.10 seconds per step using annealed importance sampling [180]. Finally, uncertainty estimation for the variant described in Section D.3 that uses $n = 5$ implicit models required 2.599 ± 0.002s. While the time complexity should grow as quadratic in n, in practice we observe that for small values of n the growth is closer to linear as the latency is dominated by the $\mathcal{O}(n)$ sampling process rather than the $\mathcal{O}(n^2)$ forward passes.

Time	IFL	IIFL
Training step (s)	0.0385 ± 0.0205	0.694 ± 0.207
Training step per model (s)	0.0077 ± 0.0041	0.347 ± 0.104
Inference (s)	0.0060 ± 0.0395	0.494 ± 0.045
Uncertainty estimation (s)	0.0029 ± 0.0008	0.988 ± 0.008

Table D.4: Computation times for training, inference, and uncertainty estimation for IFL and IIFL.

Appendix E

Appendix for Chapter 7

In Appendix E.1 and E.2, we provide implementation and hyperparameter details for all flattening and folding algorithms. In Appendix E.3, we provide information about the graphical user interface used to collect human-labeled data. In Appendix E.4, we describe the implementation details of the action primitives we use in this work. In Appendix E.5, we provide the results of ablation studies suggesting the usefulness of various design choices.

E.1 Flattening Algorithm Details

RAND

As described in the main text, this baseline simply selects the pick point p_0 uniformly at random from the garment mask. The place point p_1 is sampled uniformly at random from the workspace, but resampled if p_1 is separated from p_0 by more than 50% of the workspace in either x or y. This prevents excessively large action deltas, which has been shown to be useful in prior fabric manipulation work [100].

HUMAN

As described in the main text, this baseline allows the human to freely specify both pick and place points via the graphical user interface in Section E.3.

AEP

Our analytic smoothing policy is based on the observation that generally pulling outwards on the shirt increases its coverage over time. Specifically, we do the following:

- Compute the edge mask of the current shirt by using the formula $\texttt{xor}(\texttt{erode}(\mathbf{o}_t^m))$, where $\texttt{xor}(\cdot)$ is pixelwise exclusive or and $\texttt{erode}(\cdot)$ is a function that removes a set number of pixels (in our case, 40) at the boundary of the input mask.

- Sample uniformly among the resulting pixels to choose p_0.
- Choose the place point analytically by moving away from the center of mass and towards the edge of the shirt. Specifically, we compute p_1 as

$$p_1 = p_0 + k_1 \cdot (p_0 - \text{com}(\mathbf{o}_t^m)) + k_2 \cdot (\text{bgd}(\mathbf{o}_t^m, p_0) - p_0)$$

  where $\text{bgd}(\mathbf{o}_t^m, p_0)$ is the closest background (i.e., non-shirt) pixel to p_0 and $k_1 = k_2 = 23$ are tuned constants, but action magnitudes are shrunk to $k_1 = k_2 = 14$ for higher coverage states (at or equal to 75%) to improve stability and convergence to a flattened state.

IDYN

We first collect a dataset of 4402 random actions (90% train / 10% test split) by allowing the robot to run the RAND algorithm autonomously. The garment is reset after every 100 actions (with the procedure in Section E.4). We then train a Siamese CNN on $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$ tuples to output the $\mathbf{a}_t$ that takes $\mathbf{o}_t$ to $\mathbf{o}_{t+1}$. The convolutional layers are a ResNet-34 backbone with rectified linear unit (ReLU) output activation and pretrained (but not frozen) weights, shared between the two heads of the network. Each head encodes the input image into a vector of dimension 1,000. The two vectors are concatenated and passed through a fully connected layer with output size 2 and sigmoid activation, which generates the pick point. The predicted pick point is then concatenated with the two vectors and passed through another fully connected layer with output size 2 and sigmoid activation, which generates the pick-conditioned place point. The CNN has a total of 21.8 million parameters.

We train the network for 100 epochs with a batch size of 16 and an Adam optimizer with learning rate $1e-4$ and ℓ_2 regularization $1e-5$, saving the model weights with the lowest test loss. For stability the observation and action values are normalized to $[0, 1]$, and the 640×360 images are center cropped to 360×360. After training, the ℓ_2 distance between predicted points and ground truth points are 51.1 ± 22.8 pixels (10.0 ± 4.5% of maximum error) on the test set; see Figure E.1 for examples. At test time, the first input to the network is the current observation and the second input is an observation of the fully flattened shirt. The network output specifies the pick-and-place action to execute on the system. In this algorithm and other algorithms with model outputs, if the pick point misses the fabric, it is analytically corrected to the nearest point on the fabric mask.

CRL

Here we use the same dataset as the previous section. We train a CNN to predict the scalar change in pixel coverage $\text{cover}(\mathbf{o}_{t+1}^m) - \text{cover}(\mathbf{o}_t^m)$ from inputs $(\mathbf{o}_t, \mathbf{a}_t)$. Similar to IDYN, we use a ResNet-34 with pretrained weights to encode $\mathbf{o}_t$ into a 1000-dimensional vector. We also tile the action $250\times$ to match the dimension of the encoded image, which was found to improve performance in this work and prior work [268]. After concatenating the two

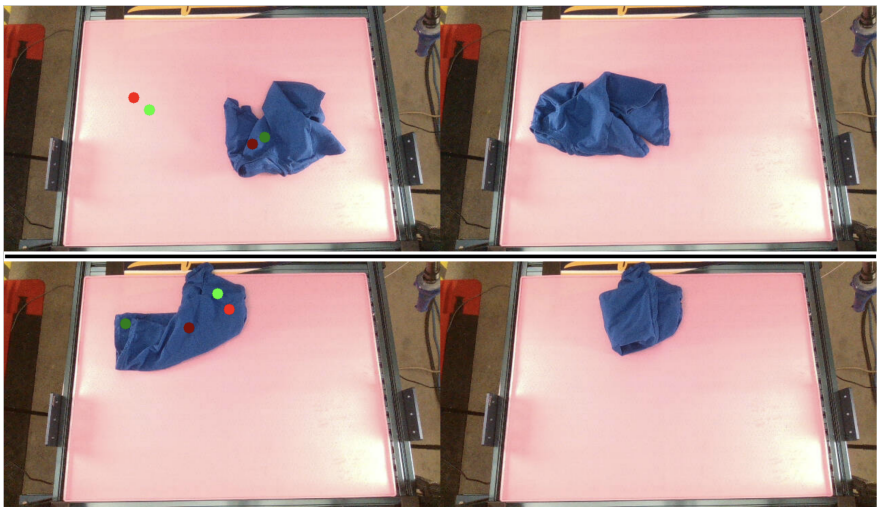


Figure E.1: IDYN model predictions on the test set. Each of the two rows is an individual transition where the left image is taken at time t and the right image is taken at time $t+1$. Ground truth pick-and-place points are in green (dark green for pick, light green for place) and predictions are in red (dark for pick, light for place). Transitions involving translations of the fabric are modeled well (top; mean ℓ_2 pixel distance 24.9) while more rare transitions such as folding are more challenging (bottom; mean ℓ_2 pixel distance 60.9).

vectors, the state is passed through two fully connected layers of size $(2000, 256)$ and $(256, 1)$ respectively with ReLU activation and hyperbolic tangent output activation to produce the predicted change in coverage. The CNN has a total of 22.3 million parameters.

As in IDYN, we train for 100 epochs with a batch size of 16 and an Adam optimizer with learning rate $1e-4$ and ℓ_2 regularization $1e-5$, saving the model weights with the lowest test loss. For stability, observation and action values are normalized to $[0, 1]$ and the delta coverage values are normalized to $[-1, 1]$. The 640×360 images are center cropped to 360×360. After training, the predicted coverage deltas are 2603 ± 2230 pixels ($2.6 \pm 2.2\%$ of maximum error) off from the ground truth labels on the test set. At test time, we randomly sample 10,000 pick-and-place actions on the current observation with the RAND strategy and select the action that results in the highest network output. The forward passes are batched into 100 observation-action pairs at a time to decrease total inference time. We also implemented action sampling with the Cross Entropy Method (CEM) but found that this did not significantly change the output actions.

$\mathbf{LP_0AP_1}$

To train the imitation learning model, we first collect a dataset of 108 human-labeled pick points on the T-shirt. The pick points are collected through the GUI in Section E.3, which shows the analytically computed place points to the human teleoperator before the actions

are executed. The dataset is augmented with online data via 3 iterations of DAgger [215], with potentially multiple pick points labeled per individual observation.

We train an Fully Convolutional Network (FCN) [233] with a ResNet-34 backbone to output a heatmap corresponding to pixel-wise pick affordances, using a sigmoid activation function at the final layer and binary cross-entropy loss. We center-crop the input images to 320×320. To provide a smooth target for the neural network, we add Gaussian distributions ($\sigma = 8$ pixels) around each labeled pick point. While training, we augment the dataset using the following sequential operations in a random order:

1. Flip left/right with probability 50%.
2. Flip up/down with probability 50%.
3. Rotate $r_k 90°$ where $r_k \sim \mathcal{U}(\{0, 1, 2, 3\})$.
4. Apply a continuous rotation of $r_k \sim \mathcal{U}(-90, 90)$.
5. Apply an affine transform with x and y scales in the range $(0.9, 1.1)$, translation percentages of 5%. .

We train for 100 epochs with a batch size of 4, using an Adam optimizer with a learning rate of $1e-4$. The total number of trainable parameters in the network is 22.8 million.

At runtime, we sample a pick point from the thresholded (value > 0.1) heatmap with sampling probability proportional to the intensity of the output pixel value. The place point is computed the same way as is done in AEP.

KP

We collect a training dataset with human-annotated keypoints for the collar, sleeve midpoints, and base corners of the shirt (when discernible). We first collect and label 305 images with the RAND policy to train an initial KP policy; we execute this policy to collect an additional 376 images and labels that better represent the distribution of encountered states. We train the same FCN as the previous subsection with the same hyperparameters but have *three* output heatmaps (instead of one) for the collar, sleeves, and base points. These heatmaps are transformed into a series of points by thresholding the normalized heatmaps with a lower bound of 0.2, with each class of points restricted to an upper-bound of 1, 2, and 2 points for the collar, sleeve, and base points, respectively.

To achieve the optimal transform between the current and template flattened T-shirt, we compute $\arg\min_{\theta} \|p_{obs} - R_\theta p_{template}\|_2$ where p_{obs} specifies the $SO(2)$ pose of the T-shirt, R_θ is the 2D rotation matrix corresponding to a rotation of θ, and $p_{template}$ specifies the $SO(2)$ pose of the template T-shirt centered at the visual center of mass of the observed t-shirt. The $\|\cdot\|_2$ cost is computed over the *visible keypoints* and their corresponding points on the template T-shirt. Using the best-match template and observation keypoints, we find

the keypoint pair with the largest ℓ_2 error and execute a pick-and-place action to move the T-shirt keypoint to its target location on the template.

DROP

The DROP algorithm is a variant of the LP_0AP_1 algorithm. Here, we attempt to intelligently combine a dynamic primitive (drop) with the algorithm to reduce the number of steps required, as follows:

1. Define $\mathcal{S}_d$ as a discretized state space of coverage values that evenly divides the possible coverage values into 200 bins. Define a hierarchical action space $\mathcal{A}_d = \{\text{LP}_0\text{AP}_1, \text{drop}\}$.

2. Model the transition dynamics of each action in $\mathcal{A}_d$. For LP_0AP_1, model

$$P(s'|s, \text{LP}_0\text{AP}_1) = s + P(\Delta s|\text{LP}_0\text{AP}_1)$$

   Boundary cases are handled via clipping and re-normalizing the distribution. For drop actions, we model

$$P(s'|s, \text{drop}) = P(s'|\text{drop})$$

   Intuitively, we model the coverage after a drop as independent of the previous state and the change in coverage from pick-and-place actions as conditional on the previous state. We estimate these quantities in a data-driven manner by executing rollouts of LP_0AP_1 or drop and recording the (s, s') tuples.

3. Once we obtain a full matrix of transition probabilities $P(s'|s, a)$, with $s, s' \in \mathcal{S}_d, a \in \mathcal{A}_d$, we run tabular Q-iteration using Bellman backups, no discount factor, and the reward function below to encourage reaching a high-coverage state as quickly as possible.

$$r(s, a) = \begin{cases} -1 & s < C \\ 0 & s \geq C \end{cases}$$

   Transition dynamics at $s \geq C$ are modified to remain stationary.

4. Once we obtain an optimal policy from Q-iteration, we run this on the robot. In practice, the policy ends up in the form

$$\pi(s) = \begin{cases} \text{drop} & s < T \\ \text{LP}_0\text{AP}_1 & s \geq T \end{cases}$$

   Intuitively, the stochastic drop can be interpreted as a geometric series with some probability of escaping the low-coverage regime, after which it is optimal to execute pick-and-place actions so as to prevent eliminating progress in high coverage states. Q-iteration returns $T = 45\%$.

E.2 Folding Algorithm Details

HUMAN

As described in the main text, this baseline allows the human to freely specify both pick and place points via the graphical user interface in Section E.3.

ASM

Analytic Shape-Matching requires only a single human demonstration. The human is shown a template image $\mathcal{T}$ of the flattened T-shirt and selects pick and place points for a folding sequence. These points are specified only once for a given folding sequence.

During execution, we compute the best match of the template with the observation $\mathbf{o}_{\mathrm{t}}^{m}$ as follows:

$$\arg\max_{\theta} \ \texttt{sum}(\texttt{and}(\mathbf{o}_t^m, R_\theta \mathcal{T} + (\mathrm{com}(\mathbf{o}_t^m) - \mathrm{com}(\mathcal{T}))))$$

where R_θ is the 2D rotation matrix corresponding to a rotation of θ, $\texttt{and}(\cdot)$ is the pixelwise AND operation, and $(\mathrm{com}(\mathbf{o}_t^m) - \mathrm{com}(\mathcal{T}))$ is a translation from the template shirt's center of mass to that of the observation. We then transform each demonstration point p with the resulting transform (i.e., $R_\theta p + (\mathrm{com}(\mathbf{o}_t^m) - \mathrm{com}(\mathcal{T}))$) to get the corresponding action. Before executing actions we project pick points and place points onto the shirt mask to avoid missed grasps.

$\mathbf{LP}_0\mathbf{LP}_1$

We collect 2 folding demonstrations with 4 pick-and-place actions each via human teleoperation. We augment the dataset by a factor of 20 to get a total of 160 data points to reduce overfitting and build robustness to rotations and translations. Specifically, we perform the following affine transforms in a random order:

1. Rotate a random multiple of 90 degrees (i.e., 0, 90, 180, or 270)
2. Scale the image in x uniformly at random between 95% and 105%
3. Scale the image in y uniformly at random between 95% and 105%
4. Rotate uniformly at random between -45 degrees and 45 degrees
5. Translate in x uniformly at random between -5% and 5%
6. Translate in y uniformly at random between -5% and 5%

We also crop the images around the visual center of mass com(·) (without crossing the workspace bounds) to encourage further translational invariance. We then train and run inference with the same FCN as LP_0AP_1 but predict 2 output keypoints instead of 1 (i.e., pick and place point). See Figure E.2 for test set predictions. At test time we execute the model outputs and terminate after 4 actions, though the termination condition may be learned in future work to allow more closed-loop behavior.

A-ASM

This baseline simply executes ASM after the flattening is performed by LP_0AP_1 instead of HUMAN.

E.3 Graphical User Interface

To enable human teleoperation, we develop a simple graphical user interface (GUI) with OpenCV (`https://opencv.org/`). The GUI displays the overhead RGB camera image of the current garment state and allows the user to specify pixels for the pick and/or place point with the mouse (Figure E.3). These points are deprojected into 3D world coordinates through the known depth camera transform and parameterize a pick-and-place action for the robot to execute. The place points are analytically computed for LP_0AP_1 and human-specified for LP_0LP_1, HUMAN flattening, and HUMAN folding. The GUI is used to provide demonstration data for the former two algorithms and execute actions for the latter two. Human labels of visible keypoints are collected with a similar interface for the KP algorithm.

E.4 Action Primitive Details

Flattening Pick-and-Place

A flattening pick-and-place primitive is parameterized by a 2D pixel pick point p_0 and a 2D pixel place point p_1. It consists of the following steps, where all steps but the first are calls to the `step()` function of the PyReach Gym API:

- Deproject the pick point p_0 into the world coordinates (x_1, y_1, z_1) of the top layer of the fabric at p_0 via the known depth camera transform. Deproject the place point p_1 into world coordinates (x_2, y_2, z_2).
- Move the gripper to a fixed initial pose in the center of the workspace about 0.2 meters above the worksurface with the gripper oriented top-down and the jaw open.
- Translate the gripper to (x_1, y_1) without changing the height or rotation.
- Lower the gripper to z_1.

Figure E.2: LP_0LP_1 keypoint predictions on the test set (i.e., unseen affine transforms). Each row is a different data sample where the left image shows the predicted pick and the right shows the predicted place. The 4 steps together comprise a folding demonstration.

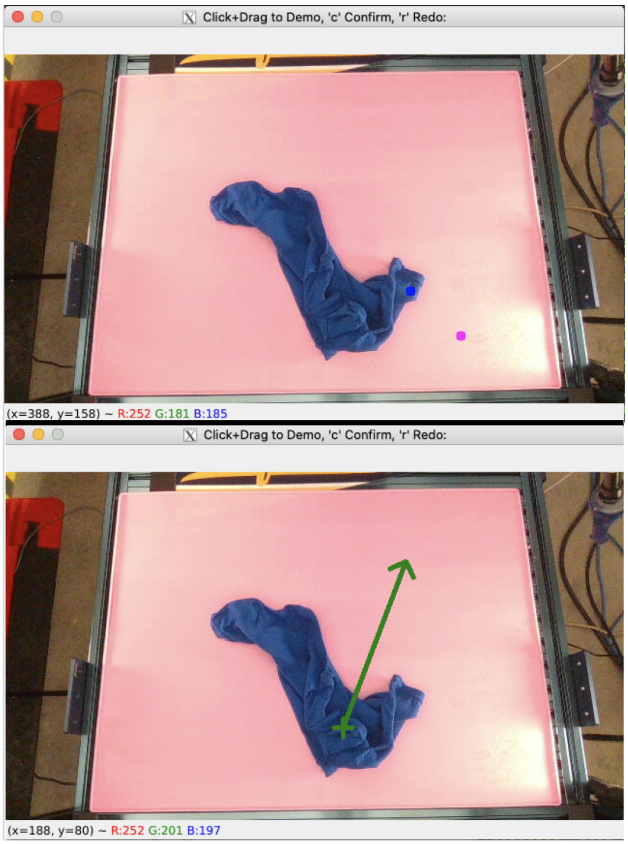


Figure E.3: **Top:** The user clicks to select a pick point and the place point is automatically computed with the strategy in Section E.1. **Bottom:** The user clicks to select a pick point, drags the mouse, and releases to select a place point.

- Close the gripper to grasp the fabric.
- Raise the gripper to 0.1 meters above its current height.
- Translate the gripper to (x_2, y_2) without changing the height.
- Open the gripper to release the fabric.
- Move the arm out of the field of view before the next image observation is captured.

All arm movements are commanded with joint velocity limits 1.04 radians per second and joint acceleration limits 1.2 radians per seconds squared.

Folding Pick-and-Place

A folding pick-and-place primitive is also parameterized by a 2D pixel pick point p_0 and a 2D pixel place point p_1. It consists of the following steps, where differences from the flattening motion are in bold:

- Deproject the pick point p_0 into the world coordinates (x_1, y_1, z_1) of the top layer of the fabric at p_0 via the known depth camera transform. Deproject the place point p_1 into world coordinates (x_2, y_2, z_2).
- Move the gripper to a fixed initial pose in the center of the workspace about 0.2 meters above the worksurface with the gripper oriented top-down and the jaw open.
- Translate the gripper to (x_1, y_1) without changing the height or rotation.
- Lower the gripper to **at most 0.1 meters below z_1, but not below the height of the worksurface.**
- Close the gripper to grasp the fabric.
- Raise the gripper to 0.1 meters above its current height.
- Translate the gripper to (x_2, y_2) without changing the height **at a lower speed than the other motions.**
- **Lower the gripper to z_1.**
- Open the gripper to release the fabric.
- **Raise the gripper to 0.1 meters above z_1.**
- Move the arm out of the field of view before the next image observation is captured.

All arm movements but the translation to (x_2, y_2) are commanded with joint velocity limits 1.04 radians per second and joint acceleration limits 1.2 radians per seconds squared. The translation has a joint velocity limit slower than the other motions by a factor of four (i.e., 0.26 radians per second).

COM Drop

A center-of-mass (COM) drop motion is parameterized by pick point p_0, which is automatically computed as the visual COM of the fabric. Differences from the flattening pick-and-place motion are in bold.

- Deproject the pick point p_0 into the world coordinates (x_1, y_1, z_1) of the top layer of the fabric at p_0 via the known depth camera transform.

- Move the gripper to a fixed initial pose in the center of the workspace about 0.2 meters above the worksurface with the gripper oriented top-down and the jaw open.
- Translate the gripper to (x_1, y_1) without changing the height or rotation.
- Lower the gripper to z_1.
- Close the gripper to grasp the fabric.
- **Move the gripper to 0.3 meters above the center of the workspace.**
- Open the gripper to release the fabric.
- Move the arm out of the field of view before the next image observation is captured.

Reset

A 'crumple' or reset operation for flattening performs the following motion 6 consecutive times:

- Move the arm to a fixed initial pose in the center of the workspace about 0.2 meters above the worksurface with the gripper oriented top-down and the jaw open.
- Compute a random x and y offset, each uniformly sampled between -0.1 and 0.1 (For reference, the workspace is about 0.7m by 0.5m.).
- Move the gripper to the center of the workspace at the height of the workspace, but offset in x and y as above.
- Close the gripper.
- Move the gripper to 0.3 meters above the center of the workspace.
- Open the gripper.

Recenter

During flattening rollouts, if the visual center of mass is sufficiently far away from the center of the workspace in x or y (by about 100 pixels in each direction, where the full image is 640×360), we perform a recentering primitive. This is a flattening pick-and-place primitive where p_0 is the pixel nearest to the center of the workspace with a small amount of noise applied ($\pm$ 12.5 pixels in x and y) and p_1 is the center of the workspace.

Recovery

We perform a random recovery action if no flattening progress is being made, i.e., coverage is below 75%, at least 5 actions have been executed, and the last 2 actions achieve lower mean coverage than the preceding 2 actions. Specifically, we select a random p_0 on the shirt mask and compute p_1 with AEP.

E.5 Ablation Studies

We run ablation studies on LP_0AP_1 to evaluate the efficacy of various components. We find that multiple design choices, when eliminated, cause the average number of actions required to flatten the shirt to increase dramatically.

Recovery Actions

For this experiment, we disable the random recovery actions we take when we detect a lack of progress in coverage (Section E.4).

Recentering Actions

Here, we remove the recentering actions we take when the shirt's visual center of mass is too far from the center of the workspace (Section E.4).

Action Shrinking

Here, we disable the reduction in action magnitudes when the shirt reaches higher coverage states (Section E.1).

The ablation experiments demonstrate that the additional primitives we introduce indeed improve the performance of LP_0AP_1. Figure E.4 and Table E.1 illustrate the slower convergence of LP_0AP_1 without each of these primitives and optimizations to a fully flattened t-shirt state. These experiments show that for practical behavior cloning involving deformable objects, adding such manipulation primitives can be beneficial in accelerating progress to the goal state.

Table E.1: Flattening ablation results. We report maximum coverage, number of actions, number of samples in the dataset, and evaluation time, where averages and standard deviations are computed over 10 trials.

Algorithm	**% Coverage**	**Actions**	**Dataset**	**Time/Act (s)**
LP_0AP_1	**97.7 ± 1.4**	**31.9 ± 17.2**	524	25.6 ± 0.9
No Recovery	96.0 ± 7.3	53.5 ± 30.3	524	25.5 ± 1.2
No Recentering	96.4 ± 1.0	65.6 ± 27.2	524	25.6 ± 1.2
No Shrinking	94.8 ± 6.6	61.2 ± 28.9	524	25.5 ± 0.8

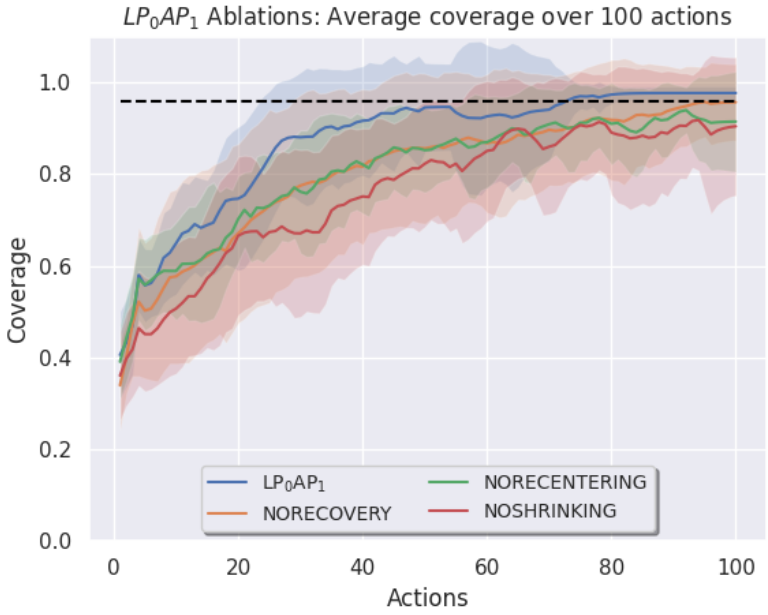


Figure E.4: Coverage vs. time for each of the ablation experiments. Shading represents one standard deviation, and the horizontal dashed line is the flattening succcess threshold (96%). All ablations converge less quickly than LP_0AP_1.