NORTHWESTERN UNIVERSITY

Single-Shot Embodied Robot Learning from Scratch

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Mechanical Engineering

By

Allison Pinosky

EVANSTON, ILLINOIS

December 2024

Abstract

Single-Shot Embodied Robot Learning from Scratch

Allison Pinosky

How can we enable robots to efficiently explore and learn in our messy real world? In this dissertation, I develop strategies to enable robots to quickly and automatically make decisions about what data to collect in complex real-world conditions to improve learned representations. The learned representations explored in this thesis can be split into two parts—Reinforcement Learning (RL) and perception learning.

For the RL part, I start by outlining a method of applying hybrid control theory to RL tasks to enable an agent to benefit from the current strengths of both model-free and model-based RL paradigms. Then, I present a method to robustly improve RL performance by decorrelating agent experiences. Finally, I propose a hardware RL benchmark for testing RL algorithms to assess their applicability to real-world systems.

For the perception learning part, I introduce a closed-loop sensor-independent embodied learning method that does not require pre-existing datasets, offline compute, or labeling to

learn latent perception models. Instead, by exploiting control and the subsequent ability to shape the data it acquires, the robot regulates the learning process through data collection, reasoning about needed data based on the evolving state of the generative model at every learning iterate. Then, I demonstrate how these learned representations of the domain can be used to separate features into distinct objects. Finally, I show that the learned perception model and the low-dimensional representations of learned objects can be used for an object identification task in new environments.

Acknowledgments

First and foremost, I would like to thank my advisor Todd Murphey. Thank you for supporting my (admittedly sometimes frustrating) insistence on running algorithms on hardware systems to believe that they really work and for helping me develop the mathematical framing and general language to communicate the behaviors I was observing during my experiments. Thank you for encouraging me to overcome my reluctance to publish work that I didn't feel was done yet and for helping me establish intermediate milestones for my projects. One of the things that drew me to your lab was the variety of projects that students work on throughout their PhD, so thank you for also supporting my collaborations with lab members on various projects—including those outside the scope of the work presented in this dissertation. Your enthusiasm, support, and mentorship are greatly appreciated.

I would also like to thank my committee members Ed Colgate, Etienne Elie, and Ryan Truby for their thoughtful feedback and for asking challenging and interesting questions. Thank you for encouraging me to practice communicating the core contributions of my work early and often. In particular, I would like to thank Etienne Elie and Intel Corporation for the hardware loans and technical support, which enabled me to explore the potential of real-time hardware-in-the-loop active learning without being limited by computation.

To my labmates, thank you for the many CRB lunches and lakefill walks—being a graduate student would have been a lot less fun without all of you. Thank you for letting me bug you about what you were working on and help envision hardware experiments for your work. In particular, I would like to thank my collaborators for challenging me to be a better researcher. To Ian, thanks for welcoming me to the lab and introducing me to the world of robot learning. Your endless questions and deep knowledge of the field continue to be inspirational to me. To Tommy, thank you for always being willing to argue with me and for bringing an unbridled enthusiasm to our work. Beyond the mathematical rigor you bring to any discussion, I appreciate your ability to envision exciting future directions on-the-fly and broaden my view of the potential impact of my work.

I'd also like to thank my family for all their support over the years. To my parents, thank you for always pushing me to be the best version of myself and for being amazing role models. Thank you for frequently texting me to check in (even when I forget to reply) and for always picking up the phone. To my brother, thanks for getting even more education than me to make sure I graduate before you. But all kidding aside, I feel so lucky to have grown up with you. And lastly, a special thank you to my husband. Thank you for uprooting your life and business to move to Chicago with me. Thank you for your continuous support and encouragement, for making sure I take breaks to go for a walk, and for generally being my biggest cheerleader. I am endlessly grateful and feel so lucky to have you by my side.

Finally, this dissertation would not have been possible without the funding sources that supported the work. In particular, this dissertation was supported by the US Office of Naval Research grant N00014-21-1-2706, the US Army Research Office grant W911NF-22-1-0286, and the McCormick School of Engineering Terminal Year Fellowship.

Abbreviations and Acronyms

i.i.d.	independent and identically distributed
AU	Active Units
CVAE	Conditional Variational Autoencoder
FDM	Fused Deposition Modeling
IMP	Integral Probability Metric
IPS	Indoor Positioning System
KL	Kullback-Leibler
MaxDiff RL	Maximum Diffusion Reinforcement Learning
MaxEnt RL	Maximum Entropy Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
NN-MPPI	Neural-Network Model Predictive Path Integral Control
PAC	Probably Approximately Correct
PAC-MDP	Probably Approximately Correct in Markov Decision Processes
ReLU	Rectifying Linear Unit
RL	Reinforcement Learning
ROS	Robot Operating System
SAC	Soft Actor Critic
SOTA	state-of-the-art
VAE	Variational Autoencoder

Table of Contents

Abstract **3**

Acknowledgments **5**

Abbreviations and Acronyms **7**

List of Tables **12**

List of Figures **13**

1 Introduction **16**

1.1 Contributions 17

1.1.1 Hybrid Control for Combining Model-Based and Model-Free RL 17

1.1.2 Decorrelating Reinforcement Learning Experiences 18

1.1.3 NoodleBot: A Hardware Reinforcement Learning Benchmark 20

1.1.4 Perception Learning for Unknown Objects with an Unknown Sensor .. 21

1.2 Dissertation Outline 23

2 Background and Related Work **24**

2.1 Reinforcement Learning 25

2.1.1 Model-Based vs Model-Free RL 25

2.1.2 Markov Decision Processes 27

2.1.3 Episode Terminology 28

2.1.4 Imitation Learning 29

2.2 Generative Modeling 30

2.2.1 Variational Autoencoders 30
2.2.2 Conditional Variational Autoencoders 33
2.2.3 Pre-trained Encoders for RL 34
2.3 PAC Learning 34
2.3.1 PAC-MDP Theory 38
2.3.2 PAC-Bayes Theory 39
2.4 Active Learning 43
2.4.1 Ergodic Control 44
2.5 Simulations and Benchmarking 46

3 Hybrid Control for Combining Model-Based and Model-Free RL 49
3.1 Hybrid Learning Approach 50
3.1.1 Deterministic Method 52
3.1.2 Stochastic Method 58
3.2 Experiments 63
3.2.1 Simulated Benchmarks 63
3.2.1.1. Related Work Benchmarking 68
3.2.2 Hardware Experiment 70
3.3 Discussion and Summary 71

4 Decorrelating Reinforcement Learning Experiences 73
4.1 Maximum Diffusion Theory 73
4.1.1 MaxDiff Generalizes MaxEnt 78
4.1.2 Single-Shot Learning 83
4.1.3 Learning Robustness 85
4.2 Methods 86
4.3 Point-mass Environment 88
4.3.1 Temporal Correlations Impact Performance 89
4.3.2 Generalization Across Initialization Locations 90
4.4 Simulated Benchmarks 92
4.4.1 Influence of the Temperature Parameter 92

4.4.2 Robustness to Initialization 93
4.4.3 Generalization Across Body Morphologies 95
4.4.4 Extension to Single-Shot Learning 96
4.5 Discussion and Summary 98

5 NoodleBot: A Hardware Reinforcement Learning Benchmark 100
5.1 Robot Design 101
5.2 Learning Environment 102
5.3 Reinforcement Learning Benchmarks 104
5.4 Results 106
5.5 Discussion 110

6 Perception Learning for Unknown Objects with an Unknown Sensor 111
6.1 Embodied CVAE Learning Theory 112
6.2 Methods 123
6.2.1 Control Formulation 124
6.2.2 Embodied CVAE Loss Formulation 126
6.2.3 Active Units 129
6.2.4 Object Fingerprinting 129
6.2.5 Object Identification 130
6.3 RGB Camera Experiments 131
6.3.1 Active Learning (Original CVAE Loss) 131
6.3.2 Object Identification (Original CVAE Loss) 134
6.3.3 Adaptive, Active Learning (New Embodied CVAE Loss) 139
6.4 Ablation Experiments 142
6.4.1 Learning 143
6.4.2 Fingerprinting 145
6.4.3 Identification 145
6.5 Experiments with Additional Sensors 149
6.5.1 GelSight Touch Sensor 149
6.5.2 Ultrasound Imager 154

6.6 Discussion and Conclusion 157

7 Conclusions **159**

7.1 Computation 159

7.2 Future Directions 161

Bibliography **163**

A Supplementary Material for Chapter 3 **183**

B Supplementary Material for Chapter 4 **192**

C Supplementary Material for Chapter 6 **196**

List of Tables

5–1 NoodleBot Details by Link 102

5–2 Swimmer Environment Parameters Comparison (Simulation vs Hardware) 104

5–3 NoodleBot Learning Hyperparameters 106

6–1 RGB Camera Latent Space Activity 133

6–2 RGB Camera Adaptive Identification Results 141

List of Figures

2–1 Multi-shot vs Single-shot 29

2–2 Active Units Illustration 32

2–3 PAC Learning Samples Example 36

2–4 PAC Learning *i.i.d.* Example 37

2–5 Ergodic Control Example 46

3–1 Deterministic Hybrid Learning Simulation Results 64

3–2 Stochastic Hybrid Learning Simulation Results 65

3–3 Comparison of Model and Policy Evolution for Stochastic Hybrid Learning 66

3–4 Comparison of Final Model-Based and Model-Free Hybrid Learning Policies 67

3–5 Hybrid Learning Comparison to Related Work Combining Model-Based and Model-Free Methods 69

3–6 Sawyer Clutter Hybrid Learning Experiment Results 70

4–1 MaxDiff Point-mass Overview 88

4–2 MaxDiff Point-mass Evaluation Paths 91

4–3 MaxDiff Swimmer Benchmark Results 94

4–4 Single-shot and Irreversible State Transitions Illustration 96

4–5 MaxDiff Simulated Benchmark Results Across Environments 97

5–1 NoodleBot Overview . . . 101
5–2 NoodleBot Learning Results . . . 107
5–3 NoodleBot Training Paths by Seed . . . 108
5–4 NoodleBot Evaluation Results . . . 109
6–1 Perception Learning Environment . . . 112
6–2 Closed-Loop Active Learning Process . . . 123
6–3 Conditional Entropy Grade Illustration . . . 127
6–4 RGB Camera CVAE Learning Distributions . . . 132
6–5 RGB Camera Latent Space Activity . . . 133
6–6 RGB Camera Representative Reconstructions . . . 134
6–7 RGB Camera Learned Fingerprints . . . 136
6–8 RGB Camera Identification (Test Environment) . . . 136
6–9 RGB Camera Identification (Plant Environment) . . . 138
6–10 RGB Camera Identification (Duck Environment) . . . 138
6–11 RGB Camera Adaptive Learning Metrics . . . 140
6–12 RGB Camera Adaptive Test Workspaces . . . 141
6–13 RGB Ablation Sample Sensor Data . . . 142
6–14 RGB Camera Ablation Learning Metrics . . . 143
6–15 RGB Camera Ablation Clustering . . . 144
6–16 RGB Camera Ablation Reconstruction Results . . . 146
6–17 RGB Camera State Ablation Results . . . 147
6–18 RGB Camera Image Ablation Results . . . 148
6–19 GelSight Sample Data . . . 150
6–20 GelSight Test Environments . . . 151
6–21 GelSight Learning Results . . . 153

6–22 Ultrasound Sample Data 154
6–23 Ultrasound Learning Results 155

Chapter 1

Introduction

Machine Learning (ML) has the potential to upend robotics as we know it. Today, we control robots with highly-specialized, manually-tuned controllers, but we can enable our robots to adapt to new, challenging environments with robust, general purpose representations—if we have the tools to create them. Current machine learning methods often rely on pristine environments and vast amounts of *cleaned* data. To realize the full potential of machine learning, we need to design our algorithms to operate in the *messy* real world. Many state-of-the-art (SOTA) methods are stuck in the virtual world, dominated by synthetic data. Simulated environments should instead be used for proof-of-concept tests or as a bootstrapping method to jump-start real-world learning. Shifting our focus to embodied ML will require reframing the role of simulation and reconsidering what capabilities we want our learned systems to provide. We should develop representations that not only generalize to never-before-seen scenarios but are also capable of continuous learning—consistently using energy to physically move in a way that guarantees improved learning.

1.1 Contributions

We want to deploy robots into novel environments and achieve high task performance. RL is a natural choice to help us move beyond the constrained framework of supervised learning, which relies on vast amounts of labeled data. Ideally, RL could enable us to seamlessly train our robots to accomplish any desired task simply by specifying a reward function. But in reality, many SOTA methods fail to live up to the full potential of RL in ways which are detailed below. In this section, we discuss how we overcome many limitations of current RL methods, which prevent us from achieving the full potential of real-world, embodied RL. The primary contributions of this dissertation are described below by chapter title.

1.1.1 Hybrid Control for Combining Model-Based and Model-Free RL

I developed and completed the simulated benchmark experiments, the Sawyer hardware experiment, related work comparisons experiments, and the experimental data analysis. The contributions in this work can be summarized as follows:

(1) a hybrid control theoretic approach to robot learning motor skills,

(2) deterministic and stochastic formulations of our hybrid learning approach,

(3) demonstrations of improved sample-efficiency and task performance over SOTA model-based [**1**, **2**] and model-free RL methods [**3**] on both simulated benchmarks and a hardware task, and

(4) demonstrations of improved task performance over RL methods that combine model-based and model-free methods [**4**–**8**].

I would like to acknowledge the contributions of my collaborators Ian Abraham and Alexander Broad. Ian and Alex developed the original hybrid learning theory. Ian developed the

RL simulation pipeline and designed the initial simulated experiments. This work on hybrid learning was published in [**9**, **10**].

RL methods often require access to vast amounts of diverse data and compute to achieve good task performance. For simulated or offline systems, these limitations may be acceptable—many seeds can be run in parallel to convergence. But for embodied systems, data collection time is often costly and rapid performance improvements are necessary. By combining model-based and model-free methods, hybrid learning enables robots to learn effective policies with far fewer environment interactions. Our experimental results suggest that we should expect RL methods to be able to rapidly learn useful policies. This represents a shift from the current paradigm of assessing SOTA algorithms, which typically run training algorithms to convergence over millions of environment interactions. Furthermore, deployed RL methods often suffer from overconfidence when testing on conditions not seen during training. Real-world systems must operate in uncertain environments and are likely to encounter unseen conditions. We show that hybrid learning improves performance during training and enables real-world deployments by leveraging the benefits of model-based and model-free methods, respectively. We also show that maintaining the full hybrid controller after training allows continued benefit from the learned predictive model and model-free policy. In the next section, we introduce an exploration strategy which improves RL methods directly and provides mathematical guarantees on the exploration properties of RL agents.

1.1.2 Decorrelating Reinforcement Learning Experiences

I helped develop the Maximum Diffusion Reinforcement Learning (MaxDiff RL) theory, with a particular focus on the extension of the main control theory to reinforcement learning. I designed and created the point-mass simulation task. I also developed and completed all

experimental analysis presented in this dissertation. The contributions of this work are the following:

(1) a sequential learning approach that exploits the statistical mechanics of ergodic processes,
(2) a point-mass simulation task that demonstrates the surprisingly high failure rate of current SOTA techniques on a trivial problem due to temporal correlations,
(3) mathematical proof that maximum entropy (MaxEnt) techniques are a special case of MaxDiff when state transition dynamics are decorrelated,
(4) mathematical and experimental results demonstrating that MaxDiff enables agents to learn continually in single-shot deployments regardless of initialization, and
(5) demonstrations of robust performance over SOTA MaxEnt techniques [**2**, **11**] across popular simulation benchmarks [**12**, **13**].

I would like to acknowledge the contributions of my collaborator Thomas Berrueta, who developed the MaxDiff control theory. The work in this chapter is a combination of original, unpublished results, as well as results previously published in [**14**].

The performance of SOTA RL methods often varies across model and environment initializations. For simulated or offline systems, these limitations may be acceptable—many seeds can be run in parallel to convergence and failed initializations can be thrown out. But for embodied systems, we want our algorithms to learn good models *every time*, regardless of the particular model initialization. MaxDiff RL ensures repeatable, robust performance regardless of model initialization. Additionally, most RL methods rely on the randomness imposed by repeated episode rollouts from different environment initializations to collect the diverse data necessary to learn to achieve good task performance. By contrast, MaxDiff

RL directly encourages state exploration. MaxDiff RL is able to overcome the temporal correlations inherent to RL and enables single-shot learning (no environment resets).

1.1.3 NoodleBot: A Hardware Reinforcement Learning Benchmark

I designed and developed NoodleBot, modified the original swimmer simulation to match the hardware configuration, and completed all hardware and simulated experimental analysis presented in this dissertation. The contributions of this work are the following:

(1) the design of a 3-link hardware swimmer benchmark for RL,

(2) a comparison between hardware and simulated swimmer benchmarks, and

(3) demonstrations of three deep RL algorithms on our benchmark.

I would like to acknowledge Thomas Berrueta and Olivia Li's help with this project. The simulated swimmer experiments from the MaxDiff RL work with Tommy were the inspiration for building the hardware swimmer. Olivia helped build the hardware, develop the firmware, and debug the RL experiments. This work has been submitted and is under review.

Many RL algorithms are not suited to real-time learning due to a lack of sample-efficiency and sensitivity to algorithmic initialization. Based on the performance of Soft Actor Critic (SAC) on the simulated swimmer, it would take approximately 12 hours for SAC to learn the task on the NoodleBot hardware swimmer and even longer to run to convergence. In addition to execution time, there is often a large performance gap between SOTA RL performance on simulated benchmarks and in real-world deployments. Even the most sophisticated simulators fail to capture the richness of real-world physics beyond the most trivial environments. Our NoodleBot swimmer captures many of the complexities inherent to physical systems and provides a hardware benchmark to assess embodied RL algorithms.

1.1.4 Perception Learning for Unknown Objects with an Unknown Sensor

The goal of this chapter is to develop sensor independent perception models with information-rich latent spaces for future tasks like identification. Therefore, I investigated the properties of the learned latent space representations and developed a clustering method that allows the latent space to be used for identification tasks. I designed and conducted hardware experiments to explore the ability of our method to learn with a variety of sensors and conditional variables. Finally, I used the Probably Approximately Correct (PAC) framework to develop theoretical results on how ergodic stability enables robust single-shot learning. The contributions of this work are the following:

(1) a method for actively learning latent features of an unknown number objects of unknown objects with an unknown sensor,

(2) theoretical results demonstrating that Conditional Variational Autoencoders (CVAEs) require a different loss function than Variational Autoencoders (VAEs),

(3) theoretical results on how ergodic stability of information states enables single-shot perception learning,

(4) experimental results demonstrating the ability of our method to generate information-rich latent spaces through active learning,

(5) experimental results demonstrating the use of spatial clustering on learned the perception models to detect objects without human-labeling,

(6) experimental results demonstrating the use of the learned perception models and the latent space features of learned objects for object identification tasks in new environments, and

(7) experimental results with an RGB camera, a touch sensor, and an ultrasound imager which demonstrate the sensor independence of our method.

I would like to acknowledge Ahalya Prabhakar's contribution to this project. Ahalya developed the initial mechanical intelligence paper that learned generative sensory models through ergodic sampling [**15**]. Ahalya's initial simulated results inspired much of the work in this chapter. I would also like to acknowledge Thomas Berrueta's contribution to this project. Our many discussions about MaxDiff RL and PAC learning helped me build the mathematical foundations of this work. Finally, I would like to acknowledge Davin Landry's help designing the sensor mounts for the hardware experiments. This work was published in [**16**] and is in preparation for a long-form submission.

RL requires access to state information and good algorithm performance relies on our ability to generate useful reward functions. RL is highly susceptible to reward hacking—where the algorithm is able to achieve high rewards without actually accomplishing the task. These issues are exacerbated when the robot must contend with sensory input rather than raw state information. While many algorithms using camera data leverage the vast amounts of pre-labeled data, our method generalizes to novel sensors without access to labeled data. Our close-loop active embodied learning framework provides a reward-free approach to learning low-dimensional environment representations with novel sensors, in novel environments without labeling. Grounded in the physical environment, these representations can enable richer reward function specifications and enable RL methods achieve improved performance on embodied tasks.

1.2 Dissertation Outline

The goal of this dissertation is to enable robots to quickly and automatically make decisions about what data to collect in complex real-world conditions to improve learned representations. The following chapters focus on different approaches to physical robot learning. In Chapter 2, we provide background and related work. In Chapter 3, we introduce a method of applying hybrid control theory to RL tasks to enable an agent to benefit from the strengths of model-free and model-based reinforcement learning paradigms. In Chapter 4, we introduce an exploration approach called MaxDiff RL to improve the robustness and performance of RL methods by regulating the data collected by physical systems. In Chapter 5, we introduce a hardware benchmark for testing RL algorithms. In Chapter 6, we introduce an embodied active learning algorithm to automate data collection to improve classification of previously unknown objects using arbitrary sensors in environments with unspecified dynamics. Finally, we conclude with a discussion of future directions in Chapter 7.

Chapter 2

Background and Related Work

Most Machine Learning (ML) operates under the train-test paradigm—where collected data is used to train a model or policy that is later used to perform a task. If actions are involved in data collection, these actions are typically chosen at random. But as the state space and the action space grow, random sampling techniques are less likely to capture or information rich areas. When actions are not chosen at random, they are almost always chosen based on attempting a particular task—represented by a reward in Reinforcement Learning (RL). These reward functions are often highly-tuned and task-specific. Can we use systematic approaches to embodied machine learning to out-perform random sampling and highly-tuned reward functions? In this work, we focus on answering this question with various embodied systems that must actively collect data for RL and perception learning.

In this chapter, we provide background material to support the rest of this dissertation. First, we introduce RL as a method of supervised learning through environment interactions via rewards, which is used in Chapters 3–5. Then, we discuss generative modeling as an unsupervised[1] learning approach to perception learning, which is used in Chapter 6. Next, we introduce the Probably Approximately Correct (PAC) learning framework to provide

[1] Conditional Variational Autoencoders (CVAEs) are sometimes referred to as a semi-supervised or self-supervised method rather than an unsupervised method.

theoretical foundations for analyzing RL algorithms and perception learning algorithms, which is used in Chapters 4 and 6. Then, we introduce active learning as a method for collecting data to train generative models, which is used in Chapter 6. Finally, we discuss simulated environments and algorithm benchmarking, which are used in Chapters 3–6.

2.1 Reinforcement Learning

RL agents learn to solve tasks by exhaustively sampling data through sequential interactions with their environments. As a result, RL agents excel in simulated environments [**17**], where high-quality data can be sampled across massively parallelized instances of a task in simulation. In the real world, however, RL agents must gather experience through embodied interactions with their environment. This imposes many challenges and constraints on the learning process that can have a negative impact on agent performance. Importantly, since sampling data during real-world deployment takes time, learning some tasks can be prohibitively costly and time consuming [**18**]—limiting the impact of RL in real-world applications.

2.1.1 Model-Based vs Model-Free RL

RL algorithms can generally be divided into two categories—model-free and model-based. Model-free methods avoid the need to model the environment or dynamics by learning a direct policy mapping between states and actions [**3**,**11**,**19**]. We refer to model-free methods as "experience-based" because policies are learned through experience. Model-based methods typically learn a dynamics model and a reward function to predict the next state and next reward given the current state and a potential next action. These outputs can be used to sequentially predict into the future the expected rewards for a set of actions. These methods

have contrasting strengths and weaknesses. Model-free approaches require significantly more data and diverse experience to learn the task than model-based methods [**20**], but model-free approaches often produce better performance. The prediction capabilities of model-based methods makes them more "sample-efficient" in solving robot learning tasks [**2**, **20**, **21**], but the models are often highly complex and can require special structure [**8**, **21**–**25**]. Can we leverage the beneficial aspects of each of these methods to enable robotic systems to rapidly learn tasks with a limited amount of experience?

Recent work addresses this question by exploring alternate ways of structuring environment models by combining probabilistic models with deterministic components [**20**]. Other work has explored using latent-space representations to reduce model complexity [**22**, **23**]. Related methods use high fidelity Gaussian Processes to create models, but these methods are limited by the amount of data that can be collected [**26**]. Finally, some researchers try to improve experience-based methods by adding exploration as part of the objective [**27**–**29**]. However, all of these approaches focus on improving either model-based or model-free methods separately rather than combining model-based planning with experience-based learning.

Methods that do combine model-based planning and experience-based learning tend to do so in stages [**8**, **30**, **31**]. First, a model is used to collect data for a task to jump-start learning. Then, supervised learning is used to update a policy [**30**,**32**] or an experience-based method is used to continue the learning from that stage [**8**]. Moreover, the model is often used as an oracle that provides labels to a base policy. The aim of these methods is to train a policy that does not rely on the model after training. While these approaches do improve learning, they do not algorithmically leverage the two learning approaches in an optimal manner, which often results in objective mismatch [**33**]. The sequential optimization solves two different problems; the model-based controller seeks to improve the objective, while the

policy seeks to match the model-based controller. In Chapter 3, we introduce a method for systematically combining model-based and model-free RL methods.

2.1.2 Markov Decision Processes

Robot RL problems are often formulated as Markov Decision Processes (MDPs). A MDP is a mathematical framework for a discrete-time stochastic process. The goal of the MDP formulation is to find a mapping from states to actions that maximizes the total reward acquired from interacting in an environment for some fixed amount of time. The MDP formulation assumes the Markov property—that the result of an action taken in a given state only depends on the current state and does not depend on the prior history. A MDP can be represented as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, p, r\}$[2] contains a set of accessible continuous states $s \in \mathcal{S}$ the robot can be in, a set of continuous bounded actions $a \in \mathcal{A}$ that a robot may take, rewards r, and a transition probability $p(s_{t+1}|s_t, a_t)$. For stochastic actions, the transition probability represents the probability of transitioning from one state s_t to the next s_{t+1} given an action a_t applied at time t. For deterministic actions, the transition probability specifies a fixed next state s_{t+1} given state s_t and action a_t. With the MDP framework, the total reward goal can be written as the objective

$$\pi^\star = \arg\max_{\pi} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[\sum_{t=0}^{T-1} r(s_t, a_t) \right] \quad \text{subject to} \quad p(s_{t+1}|s_t, a_t), \tag{2.1}$$

where $\mathbb{E}$ is the expectation operator, T is the discrete time horizon, and the solution is an optimal policy $\pi^\star$. During training, RL agents must balance exploration vs exploitation when collecting training data. The MDP objective in (2.1) is often augmented with an additional term to encourage exploration during training.

[2] MDPs can also include discount factor $\gamma \in [0, 1)$, which modifies (2.1) by multiplying the reward by γ^t. Partially-Observable MDPs (POMDPs) also include observation space $\mathcal{O}$ in the representation.

Model-free approaches learn a stochastic policy $a \sim \pi(\cdot|s)$ that maximizes the reward r at a state s. Model-based approaches solve the MDP problem by modeling a transition function $s_{t+1} = f(s_t, a_t)$ and a reward function $r_t = r(s_t, a_t)$. Model-based methods either use these functions to construct a policy or directly generate actions through model-based planning [**20**]. When the transition model and the reward function are known, the MDP formulation becomes an optimal control problem. We can use any set of existing methods [**34**, **35**] to solve for the best set of actions (or policy) to maximize the reward.[3] In Chapters 3 and 4, our RL approaches are built on the MDP formulation.

2.1.3 Episode Terminology

Most RL frameworks organize an agent's interactions with the environment into episodes,[4] which consist of a sequence of steps. Episodes usually have a fixed maximum number of steps, but an episode can be stopped early if a termination condition is satisfied. Each episode typically starts with a random instantiation of the task and environment. This randomization passively provides a kind of variability that is essential to the RL process [**36**]. In Chapter 4, we refer to episodic learning as *multi-shot* learning.

However, episodic problems of this kind are very rare in real-world scenarios. When agents are deployed in the real world, they face situations at test time that were never encountered during training. Since exhaustively accounting for every possible scenario is infeasible, agents capable of real-time adaptation and learning during individual deployments

[3] Optimal control problems are often specified to minimize a cost instead of maximizing a reward; however, the analysis remains the same.

[4] We note that episode and epoch are sometimes used interchangeably in the literature, but for offline learning methods, an epoch can instead describe one training pass through all currently collected data. Therefore, we choose to use the term episode here.

are desirable [**18**]. There is a need for methods that allow agents to perform a task successfully within a single trial—what we refer to as *single-shot* learning. This concept is in line with the *single-life* RL setting considered in [**37**]. The difference between *multi-shot* and *single-shot* is illustrated in Fig. 2–1.

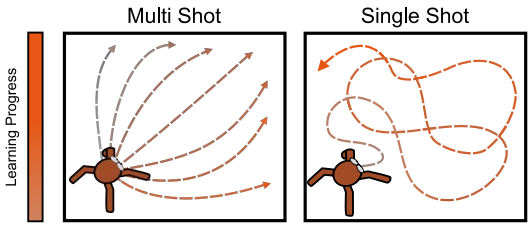

Figure 2–1 Multi-shot vs Single-shot. Typically, RL algorithms learn across many different deployments of an agent, which we refer to as multi-shot learning. Single-shot learning requires learning through a single, continuous task attempt.

Single-shot learning is distinct from *zero-shot* and *few-shot* learning, which are terms used to describe the ability of learning algorithms to generalize pre-trained models to new tasks, new classes of data, or new environments. They are also used to describe the transfer of models trained in simulation to real-world scenarios—called sim-to-real transfer. *Zero-shot* learning methods do not get access to any new training data [**38**–**41**]. *Few-shot* learning methods receive a small amount of new data [**42**, **43**] and are also referred to as *fine-tuning* [**44**, **45**]. Finally, *single-shot* is also distinct from *reset-free* learning, which describes a method of training multiple tasks at once to prevent the need for manual resetting of the environment—particularly for hardware experiments [**36**, **46**, **47**]. These methods still learn in an episodic fashion, but they switch learning tasks at the start of each episode, which naturally randomizes each episode initialization. Rather than relying on externals sources of randomization, we systematically generate diverse robot experiences in Chapters 4 and 6.

2.1.4 Imitation Learning

Imitation learning is an extension of RL, where the algorithm does not have access to the reward function. Instead, imitation learning methods learn policies based on state-action pairs from expert demonstrations. Imitation learning assumes the expert demonstrations

implicitly encode the reward function. Popular methods include behavior cloning [**48**–**50**], data aggregation (DAgger) [**51**, **52**], and inverse RL [**53**, **54**]. Although these methods do not require access to a reward function, performance is highly dependent on the expert demonstrations provided to the algorithm.

2.2 Generative Modeling

Can we learn representation without a reward function? One way to do so is by learning generative models for perception. Popular methods include sensor-specific methods like NeRF [**55**], data compression techniques like autoencoders [**56**–**58**] and diffusion models [**59**], and methods that learn to discriminate between real and fake data like generative adversarial nets (GANs) [**60**]. These generative models assume they have access to independent and identically distributed (*i.i.d.*) training data. In Section 2.4, we will discuss methods of collecting that data online, but for now, assume the models have access to training data. In this section, we introduce autoencoders as our generative method for perception learning.

2.2.1 Variational Autoencoders

A Variational Autoencoder (VAE) is a directed graphical model that learns a low-dimensional latent space in an unsupervised fashion by training two networks—an encoder and a decoder. The encoder compresses input data into a low-dimensional parameterized latent space distribution, and the decoder generates data predictions from the sampled latent space [**56**,**57**]. The primary objective of a VAE is to maximize the variational evidence lower

bound (ELBO) on the marginal log-likelihood. The objective is typically formulated as

$$\mathcal{L}(\theta, \phi; \mathbf{y}) = \underbrace{\mathbb{E}_{q_\phi(z|y)}[\log p_\theta(y|z)]}_{\text{reconstruction loss}} - \beta \underbrace{D_{KL}(q_\phi(z|y) \parallel p(z))}_{\text{latent space regularization}} \tag{2.2}$$

where y are sensor data (*e.g.*, an image), $q_\phi(z|y)$ is the latent space distribution represented by encoder parameters ϕ, $p_\theta(y|z)$ is the decoder network parameterized by θ, $p(z)$ is the prior latent space distribution, β is a hyperparameter, $\mathbb{E}(\cdot)$ represents the expectation, and $D_{KL}(\cdot\|\cdot)$ is the Kullback-Leibler (KL) divergence. To facilitate backpropagation, VAEs employ the reparameterization trick. This technique introduces an auxiliary variable $\epsilon \in \mathcal{N}(0, 1)$ to allow the random latent variable z to be expressed as a deterministic variable $z = \mu_z + \sigma_z \epsilon$, where μ_z and σ_z are parameters of the posterior distribution computed by the encoder.

A key assumption of the VAE objective is that it is optimized over a complete dataset that contains *i.i.d.* samples.[5] Two behaviors often observed during VAE training are learning-lag and posterior collapse [**61**–**64**]. Learning lag refers to delay between learning the two loss terms, as the latent space often lags behind the reconstruction loss. Posterior collapse occurs when the latent space collapses to the prior. When full posterior collapse occurs, the VAE can learn a good generative model of the data (*i.e.*, good decoder) but fail to learn good representations of the individual data points (*i.e.*, poor encoded latent space). Prior work seeks to mitigate learning-lag and posterior collapse with batch normalization and β annealing [**65**–**67**]. The tuning of these terms is sensitive to the learning dataset. VAEs combine the principles of deep learning and probabilistic modeling, offering a powerful tool for understanding and representing complex data.

[5] For mini-batch training, it is also assumed that data are drawn *i.i.d.* from the dataset.

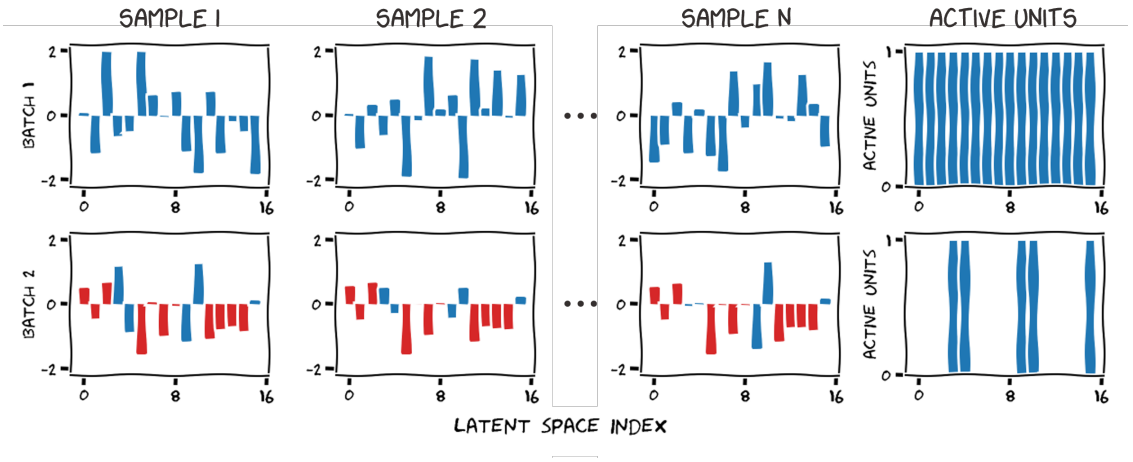

Figure 2–2 Active Units Illustration. Each row represents a different batch of data. The last column displays the active units for each row. Blue indicates active units in the last column (AU=1), and red indicates collapsed units in the last column (AU=0).

Active Units. One way to determine if latent space collapse has occurred is using the Active Units (AU) method from [**68**]:

$$\text{AU} = \sum_{l=1}^{L} \mathbf{1}[\text{Var}_{\mathcal{S}}(\mu_{(z,l)}) > \tau], \tag{2.3}$$

where L is the size of the latent space, μ_z are the latent space means, $\mathcal{S}$ a subset of the collected data points, Var$(\cdot)$ is the variance across $\mathcal{S}$, $\mathbf{1}[\cdot]$ is an indicator function, and τ is a threshold. When a latent space fully collapses, the latent space contains no active units. The larger the number of active units, the more latent space dimensions are being used by the decoder. Because active units is a measure of variance, we can need a *set* of samples to evaluate latent space activation. Fig. 2–2 illustrates active units evaluated for two different batches of data. In the top row, the variance across samples results in all latent space dimensions containing active units. In the bottom row, the latent space dimensions depicted in red have variances below the specified threshold, and therefore, those units have collapsed.

2.2.2 Conditional Variational Autoencoders

The Conditional Variational Autoencoder (CVAE) formulation augments the general VAE network architecture by adding conditional variables as inputs to both the encoder and decoder networks [**15**, **58**]. CVAEs are typically trained with the objective:

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y}) = \underbrace{\mathbb{E}_{q_\phi(z|x,y)}[\log p_\theta(y|z,x)]}_{\text{reconstruction loss}} - \beta \underbrace{D_{KL}(q_\phi(z|x,y) \parallel p(z))}_{\text{latent space regularization}} \tag{2.4}$$

where y are sensor data (*e.g.*, an image), x are conditional variables (*e.g.*, numeral label or robot state), $q_\phi(z|x,y)$ is the latent space distribution represented by encoder parameters ϕ, and $p_\theta(y|z,x)$ is the decoder parameterized by θ [**56**].

Similar to VAEs, CVAEs are susceptible to learning-lag and posterior collapse. The addition of conditional variables to the CVAE decoder means full posterior collapse can result in the decoder network learning a direct mapping from conditional variables to predicted sensor outputs. The benefit of using a VAE is that the latent space models the correlations between samples. If the latent space fully collapses, then we lose access to these correlations. Therefore, considering the quality of the data collection process is necessary to prevent catastrophic posterior collapse, particularly in applications requiring real-time data collection.

In the embodied context, CVAEs can connect the evolution of the sensor data to changes in the world and allow a common framework to be used across sensor modalities (*e.g.*, cameras, touch sensors, etc.). In a basic embodied example, the conditional variables could be the pose of the robot (*e.g.*, the position of the sensor), but in more complex scenarios, the conditional variables could also include both controlled variables (*e.g.*, position, speed, etc.) and uncontrolled variables (*e.g.*, turbulence, lighting, etc.). For instance, in a novel lighting condition—where the lighting could be parameterized by the conditional variable of ambient brightness—the robot may identify an opportunity in completely re-exploring

an object under the new lighting condition, even without being able to control lighting. Generative models give the robot a more holistic understanding of the dynamic, physical world rather than assuming the robot operates in a closed system. In Chapter 6, we use a CVAE as our learning representation for embodied perception learning.

2.2.3 Pre-trained Encoders for RL

Visual-based RL is a class of RL problems that seek to learn tasks directly from images rather than from environment states. For a robot locomotion task (*e.g.*, the ant shown in Fig. 2–1), the standard RL problem learning states might include joint angles/velocities as well as position and orientation in the world frame. The visual-based RL learning state might only include a top-down view of the robot from a fixed frame. Many state-of-the art RL algorithms can be adapted to visual-based RL tasks with the addition of an encoder network to pre-process the state and reduce its dimension. A critical issue with these adaptations is their lack of sample efficiency. In the computer vision community, there are many pretrained vision models available like ResNet and ImageNet [**69**, **70**]. To address the issue of sample efficiency, recent research has sought to leverage these pre-trained vision models [**71**–**79**]. In Chapter 6, we learn generative perception models from scratch in novel environments, but this recent work suggests that our learned perception models could be applied to RL tasks.

2.3 PAC Learning

As deep RL and generative methods have grown in popularity, there have been an increasing number of papers that seek to understand the empirical success of these methods.

Many works use the PAC learning framework as a baseline for establishing theoretical guarantees under *i.i.d.* data assumptions [**80**–**84**]. The formal definition of PAC learnability is defined in terms of the generalization error (*i.e.*, the risk associated with a hypothesis) [**85**].

Definition 2.1 (Generalization Error) *Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and an underlying data distribution $\mathcal{D}$, generalization error or risk of h is defined as*

$$R(h) = P_{x\sim\mathcal{D}}[h(x) \neq c(x)] = E_{x\sim\mathcal{D}}\left[\mathbf{1}\left[h(x) \neq c(x)\right]\right], \tag{2.5}$$

where $\mathbf{1}[\cdot]$ denotes an indicator function that evaluates to 1 when the condition in the brackets is met and is 0 otherwise.

Algorithms do not have access to the true underlying distribution, but the learner can evaluate the empirical error for a set of samples [**85**].

Definition 2.2 (Empirical Error) *Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and a sample $S = (x_1, x_2, \ldots x_m)$, empirical error or empirical risk of h is defined as*

$$\hat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\left[h(x) \neq c(x)\right]. \tag{2.6}$$

With these two error definitions, we can introduce the definition of PAC-learning [**85**].

Definition 2.3 (PAC-learning) *A concept class $\mathcal{C}$ is said to be PAC-learnable if there exists an algorithm $\mathcal{A}$ and a polynomial function poly$(\cdot,\cdot)$ such that for any $\epsilon > 0$ and $\delta > 0$, for all data distributions $\mathcal{D}$ on $\mathcal{X}$ and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $N \geq$ poly$(1/\epsilon, 1/\delta)$:*

$$P_{S\sim D^m}[R(h_s) \leq \epsilon] \geq 1 - \delta. \tag{2.7}$$

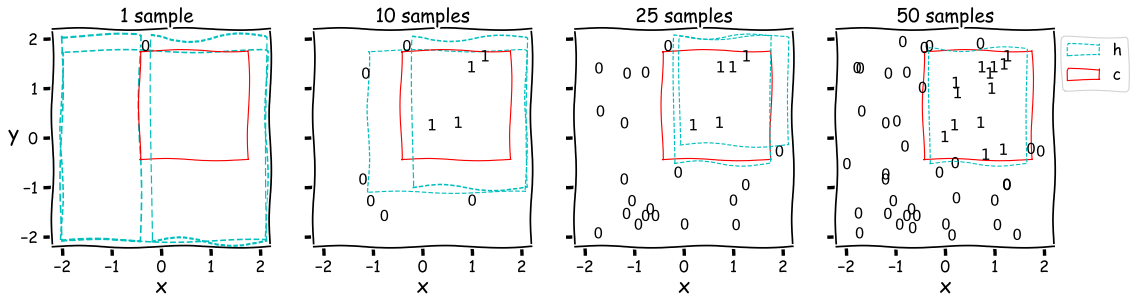

Figure 2–3 PAC Learning Example: Binary Classification in a Planar Domain. The red box shows the true underlying concept. The blue boxes show hypotheses for the provided samples. Each column shows a different number of samples. As the number of samples increases, the error decreases.

In other words, a class of functions is PAC-learnable if an algorithm can produce a function that recreates the input-output mapping of an arbitrary target function with high probability (at least $1 - \delta$) and low error (at most ϵ) after observing a number of points.

As a simple example, let's consider the task of learning to correctly label a set points in a plane. Our concept class $\mathcal{C}$ could be the set of all points inside a fixed box in the plane. In Fig. 2–3, each blue box in is a possible hypothesis h, and the red box is the unknown target concept c. Then, as we draw samples, the target concept can be used to label the samples: $c(x) = 1$ for all points inside the red target concept box and $c(x) = 0$ otherwise. As more samples are provided to the algorithm, certain hypotheses are eliminated as the sampled points violate the bounds of the boxes. Eventually, a single box is left, which correctly separates the 0 samples from the 1 samples. A learning algorithm is successful if it finds a hypothesis h that matches the target concept c. Importantly, the algorithm does not know the underlying function, so the final hypothesis may not exactly match the target concept. The ϵ term in the PAC-learning definition bounds how much the hypothesis can differ from the target concept.

A crucial assumption underlying the concept of PAC-learnability is access to *i.i.d.* data samples. Ideally, an agent (or algorithm) would exhaustively sample all regions of the data

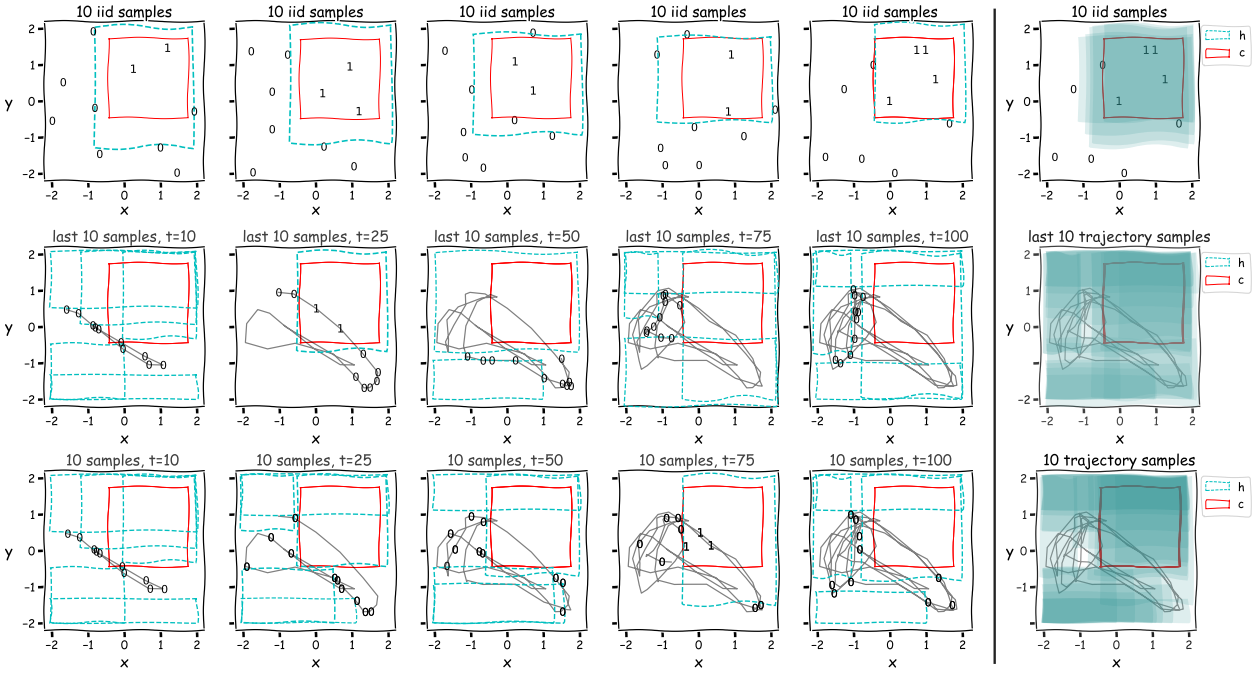

Figure 2–4 PAC Learning *i.i.d.* Example: Binary Classification in a Planar Domain. Each row shows a different sampling condition. The first row shows 10 *i.i.d.* samples from the underlying distribution. The second row shows the last 10 samples from a trajectory. The last row shows 10 samples from the same trajectory in the second row. Each of the first 5 columns shows a different set of 10 samples. The last row shows the overlapping concepts from the first columns. The last column illustrates that *i.i.d.* samples produce hypotheses with similar error bounds (compared to the true underlying concept), while the trajectory samples produce hypotheses with a large variety of errors.

distribution $\mathcal{D}$ simultaneously and produce a hypothesis from this spatial ensemble of data samples, as was done in our simple example from Fig. 2–3. To illustrate the importance of *i.i.d.* data, let's focus on the case of 10 samples from our simple planar example. In Fig. 2–3, we looked at a single set of samples. The top row of Fig. 2–4 shows five different sets of *i.i.d.* samples. The last plot in the first row shows the overlapping hypotheses for each set of samples. Although there are differences in the hypotheses generated for each set of samples, each shaded hypotheses region contains the true concept and has a similar amount of error in comparison to the true concept. This illustrates the ϵ part of the PAC bounds. The next two rows show data collected from continuous robot trajectories. Even though each plot contains 10 data points, these samples are no longer *i.i.d.* and result in vastly different

hypotheses depending on the data samples. Furthermore, these hypotheses do not overlap like the *i.i.d.* samples, illustrating how the PAC bounds do not hold for non-*i.i.d.* data. One way robot algorithms circumvent this issue is with big data, but we want our algorithms to be sample efficient. In Section 2.4, we introduce ergodic coverage via active learning as a method for collecting *i.i.d.* data.

Many algorithms and test environments require extensions to this basic PAC learning framework, two of which are discussed in the following subsections. Probably Approximately Correct in Markov Decision Processes (PAC-MDP) extends the PAC learning framework to reinforcement learning algorithms, while PAC-Bayes extends the framework to algorithms with distributions like generative models.

2.3.1 PAC-MDP Theory

PAC-MDP extends PAC learning theory to algorithms whose sample complexity can be bounded by a polynomial in the environment size and approximation parameters, with high probability [**86**]. As was introduced in Section 2.1.2, a MDP can be represented by $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, r, p\}$, where $\mathcal{S}, \mathcal{A}$ are the state space and action space. Therefore, the PAC-MDP framework needs to incorporate the sample complexity of the state and action spaces into its algorithm definition. PAC-MDP uses the following definition from [**87**]:

Definition 2.4 (MDP Algorithm Sample Complexity) *Let $c = (s_1, a_1, r_1, s_2, a_2, r_2, \dots)$ be a random path generated by executing an algorithm $\mathcal{A}$ in an MDP $\mathcal{M}$. For any fixed $\epsilon > 0$, the sample complexity of exploration of $\mathcal{A}$ is the number of time steps t such that the policy at time $t, \mathcal{A}_t$, satisfies*

$$V^{\mathcal{A}_t} < V^*(s_t) - \epsilon, \tag{2.8}$$

where $V = \mathbb{E}\left[\sum_{t=0}^{T-1} r(s_t)\right]$ *and the optimal policy has value function* V^*.

The policy of any algorithm $\mathscr{A}$ at a fixed instance in time t is defined as a function $\mathscr{A}_t : \{\mathcal{S} \times \mathcal{A}\}^* \times \mathcal{S} \to \mathcal{A}$ that maps future paths to future actions. With this definition, we can now define PAC-MDP as in [**86**, **88**]:

Definition 2.5 (PAC-MDP) *An algorithm* $\mathscr{A}$ *is said to be a PAC-MDP algorithm if, for any* $\epsilon > 0$) *and* $(0 < \delta < 1)$ *sample complexity of* $\mathscr{A}$ *is less than some polynomial in the relevant quantities* $(\mathcal{S}, \mathcal{A}, 1/\epsilon, 1/\delta)$*, with probability at least* $(1 - \delta)$.

In Chapter 4, we use the PAC-MDP framework provide representation-agnostic guarantees for our Maximum Diffusion Reinforcement Learning (MaxDiff RL) method.

2.3.2 PAC-Bayes Theory

PAC-Bayes extends PAC learning theory to probability distributions over a class of hypotheses and provides an upper bound on the model's empirical risk and its population class [**89**, **90**]. PAC-Bayes replaces the PAC error definitions (Definitions 2.1 and 2.2) with the following definition of empirical risk and true risk.

Definition 2.6 (Empirical Risk and True Risk) *Given a real-valued loss function* $\ell : \mathcal{H} \times \mathcal{X} \to [0, \infty)$*, the empirical and true risks of a posterior* $q \in \mathcal{M}_+^1(\mathcal{H})$ *is*

$$\hat{\mathcal{R}}_S(q) = \mathop{\mathbb{E}}_{h \sim q}\left[\frac{1}{n}\sum_{i=1}^{n} \ell(h, x_i)\right] \quad \textit{and} \quad \mathcal{R}(q) = \mathop{\mathbb{E}}_{h \sim q}\left[\mathop{\mathbb{E}}_{x \sim \pi} \ell(h, x_i)\right] \tag{2.9}$$

where $\mathcal{H}$ *is a hypothesis class,* $S = \{x_1, \ldots, x_n\}$ *is an* i.i.d. *sampled training set from an unknown distribution* π *over instance space* $\mathcal{X}$*, and* $\mathcal{M}_+^1(\mathcal{H})$ *is a set of probability measures on a space* $\mathcal{H}$.

This definition replaces the indicator function from Definition 2.1 with a loss function and allows us to reason over algorithms that output a distribution of hypotheses rather than a single hypothesis. The PAC-Bayes bounds are dependent on the empirical performance of q and its closeness to a chosen prior distribution $p \in \mathcal{M}_+^1(\mathcal{H})$. The following theorem states the PAC-Bayes bounds.

Theorem 2.1 (PAC-Bayes Bound) *Given a probability measure μ on $\mathcal{X}$, a hypothesis class $\mathcal{H}$, a prior distribution p on $\mathcal{H}$, a loss function $\ell : \mathcal{H} \times \mathcal{X} \to [0, \infty)$, real numbers $\delta \in (0, 1)$ and $\lambda > 0$, with probability at least $1 - \delta$ over the random draw of $S \overset{iid}{\sim} \mu$, the following holds for any posterior $q \in \mathcal{M}_+^1(\mathcal{H})$:*

$$\mathcal{R}(q) \leq \hat{\mathcal{R}}_S(q) + \frac{\lambda}{8n} + \frac{1}{\lambda}\left(D_{KL}(q\|p) + \log\frac{1}{\delta}\right), \tag{2.10}$$

where D_{KL} indicates the KL divergence.

An extension of PAC-Bayes to conditional priors was presented in [**84**]. Incorporating conditional priors into the PAC-Bayes framework uses the following assumption:

Assumption 2.1 (Conditional Posterior and Loss Function Properties) *A distribution $y \mapsto q(\cdot|y)$ and a loss function ℓ satisfy Assumption 2.1 with constant $K > 0$ if there exists a family $\mathcal{E}$ of functions $\mathcal{H} \to \mathbb{R}$ such that the following properties hold*

(1) The function $y \mapsto q(\cdot|y)$ is continuous in the sense that for any $y_1, y_2 \in \mathcal{Y}$,

$$d_{\mathcal{E}}(q(h|y_1), q(h|y_2)) \leq K d(y_1, y_2), \tag{2.11}$$

*where $d_{\mathcal{E}}$ are Integral Probability Metrics (IMPs)—see [**91**]—defined on the family of functions $\mathcal{E}$ and d is an underlying metric on $\mathcal{Y}$.*

(2) The function $\ell(\cdot, y) : \mathcal{H} \to \mathbb{R}$ is in $\mathcal{E}$ for any $y \in \mathcal{Y}$.

In [**84**], conditional priors were introduced in the context of VAEs. Incorporating the VAE networks into Assumption 2.1 requires functional representations of the encoder and decoder VAE networks. These functions as well as the loss function are defined as follows:

Definition 2.7 (VAE Encoder, Decoder, and Loss Functions) *Given a Euclidean instance space $\mathcal{Y}$ and a latent space $\mathcal{Z} = \mathbb{R}^d$, let $q_\phi(z|y)$ be a Gaussian latent space distribution $\mathcal{N}\left(\mu_\phi(y), diag\left(\sigma_\phi^2(y)\right)\right)$, where $\mu_\phi(y) : \mathcal{Y} \to \mathbb{R}^d$ and $\sigma_\phi(y) : \mathcal{Y} \to \mathbb{R}^d_{\geq 0}$. We define the encoder, decoder, and loss functions as*

$$Q_\phi(y) : \mathcal{Y} \to \mathbb{R}^{2d}, \text{ where } Q_\phi(y) = \begin{bmatrix} \mu_\phi(y) \\ \sigma_\phi(y) \end{bmatrix}, \tag{2.12}$$

$$g_\theta(z) : \mathcal{Z} \to \mathcal{Y}, \text{ where latent space samples } z \sim q_\phi(z|y), \text{ and} \tag{2.13}$$

$$\ell^\theta(z, y) : \mathcal{Z} \times \mathcal{Y} \to [0, \infty), \text{ where } \ell^\theta(z, y) = \|y - g_\theta(z)\|, \tag{2.14}$$

where $\|\cdot\|$ denotes the L_2 norm.

To satisfy the first part of Assumption 2.1, we require an assumption that both VAE networks are continuous. Using the VAE definitions, we impose the following assumption:

Assumption 2.2 (VAE Encoder and Decoder Properties) *The encoder and decoder functions are Lipschitz-continuous w.r.t. their inputs meaning there exist real numbers $K_\phi, K_\theta > 0$ such that for any $y_1, y_2 \in \mathcal{Y}$ and $z_1, z_2 \in \mathcal{Z}$,*

$$\|Q_\phi(y_1) - Q_\phi(y_2)\| \leq K_\phi \|y_1 - y_2\| \tag{2.15}$$

and

$$\|g_\theta(z_1) - g_\theta(z_2)\| \leq K_\theta \|z_1 - z_2\|, \tag{2.16}$$

where $\|\cdot\|$ denotes the L_2 norm.

We require a further statement to show that for the VAE loss function $\ell^\theta(z, y)$ there is a family $\mathcal{E}$ for which the continuity assumption is satisfied with constant K. Thus, we get the following lemma.

Lemma 2.1 (VAE Satisfies Assumption 2.1) *For a VAE with parameters ϕ and θ, let $K_\phi, K_\theta \in \mathbb{R}$ be the Lipschitz norms of the encoder and decoder respectively. Then the variational distribution $q_\phi(z|y)$ satisfies Assumption 2.1 with*

$$d_{\mathcal{E}}(q_\phi(z|y_1), q_\phi(z|y_2)) \leq K_\phi K_\theta \left(\|x_1 - x_2\| + \|y_1 - y_2\|\right) \tag{2.17}$$

and

$$\ell^\theta(z, y) \in \mathcal{E} \text{ for any } y \in \mathcal{Y}, \tag{2.18}$$

where $\mathcal{E} = \{f : \mathcal{Z} \to \mathbb{R} \text{ s.t. } \|f\|_{Lip} \leq K_\theta\}$.

Finally, we can formulate a PAC-Bayes bound for the VAE reconstruction loss, which has a similar form to Theorem 2.1.

Theorem 2.2 (VAE PAC-Bayes Bounds) *Let $\mathcal{Y}$ be the instance space, $\zeta \in \mathcal{M}_+^1(\mathcal{Y})$ the data-generating distribution, Z the latent space, $p(z) \in \mathcal{M}_+^1(\mathcal{Z})$ the prior distribution on the latent space, θ the decoder parameters, and $\delta \in (0, 1)$, $\lambda > 0$ be real numbers. With probability at least $1 - \delta$ over $S \overset{iid}{\sim} \zeta$, the following holds for any posterior $q_\phi(z|y)$:*

$$\begin{aligned} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim q_\phi(z|y)} \ell^\theta(z, y) \leq \frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q_\phi(z|y_i)} \left[\ell^\theta(z, y_i)\right] + \frac{1}{\lambda} \Bigg[\sum_{i=1}^{n} D_{KL}(q_\phi(z|y_i) \| p(z)) + \log \frac{1}{\delta} \\ + \frac{\lambda K_\phi K_\theta}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{y\sim\zeta} \left[d(y, y_i)\right] + n \log \mathop{\mathbb{E}}_{z\sim p(z)} \mathop{\mathbb{E}}_{y\sim\zeta} \exp\left(\frac{\lambda}{n} \left(\mathop{\mathbb{E}}_{y'\sim\zeta} \left[\ell^\theta(z, y')\right] - \ell^\theta(z, y)\right)\right) \Bigg], \end{aligned} \tag{2.19}$$

where K_ϕ, K_θ are encoder and decoder Lipschitz norms and $d(y, y') = \|y - y'\|$.

Similar to the VAE loss function in (2.2), these bounds assume *i.i.d.* data. In Chapter 6, we use the PAC-Bayes framework to extend the VAE PAC-Bayes bounds to embodied agents with CVAE learning representations.

2.4 Active Learning

A key challenge for embodied learning agents is the need to collect training data. Active learning is a ML strategy in which information about the model or the collected data are used to guide future data collection [**92**–**97**]. Active learning can result in lower losses or greater data efficiency relative to conventional training strategies. In deep learning, active learning can improve training time by biasing the data set or reduce data labeling costs by prioritizing certain samples for manual review. In a robotics context, active learning can guide the behavior of a system to maximize model learning. Active learning is distinct from data augmentation techniques, which seek to prevent model overfitting and increase the size of the training set by performing transformations on the existing data [**98**–**103**]. Data augmentation techniques can be used in addition to active learning, but we do not apply these techniques in the work presented in this dissertation.

Data collection is a vital part of a robot's learning process because the data influence what is learned. By framing the data collection process as an exploration problem, we can examine exploration strategies from prior work in active sensing. Active sensing [**104**–**106**] investigates how controlling sensor parameters can be used to acquire information or reduce uncertainty. The applications of this extensive literature include prioritized decision making [**107**, **108**], inspection [**109**], mine detection [**104**], object recognition/classification [**110**–**112**], next-best-view problems [**113**–**118**], and environmental modeling [**119**–**121**]. Planning

for exploration (*i.e.*, data collection) is challenging because the planning step depends not only on the sensor, but also the entity being estimated and the dynamic evolution of that entity. Thus, exhaustively searching for an action that provides an optimally informative next state is a (prohibitively) computationally intensive process [**106**,**122**–**126**]. Other active sensing work has explored using "expected information gain" to select an optimal control action based on a local estimate of the expected information [**113**, **127**–**134**]. Often, these methods do not (or cannot) incorporate general sensor dynamics [**113**,**131**–**134**]. Even global strategies are likely to suffer when uncertainty is high and information is diffuse [**135**–**137**]—exactly the scenarios we are most interested in testing.

2.4.1 Ergodic Control

In Chapter 6, we use ergodic control as our active learning method. The information metric we use to capture how much independent information is encoded in a trajectory is a measure of *ergodicity*. A trajectory $x(t) \in \mathcal{X}$ is *ergodic* with respect to a probability distribution function (PDF) $h(x)$ if the amount of time spent in any neighborhood $\mathcal{N} \subset \mathcal{X}$ is proportional to the measure of that neighborhood given by $h(x)$. Hence, an ergodic trajectory will spend more time in areas of high probability of information rich measurements and less time in areas of low probability of information rich measurements. Perfect ergodicity is only possible on infinite time horizons, leading to the need for a measure that enables *maximizing* ergodicity through decisions. Such a measure, based on the spatial Fourier transform, was developed in [**138**], and prior work uses this measure to synthesize maximally ergodic trajectories for general nonlinear systems in real-time [**139**–**141**]. In contrast to maximizing

ergodicity, entropy minimization exploits local structure in a probability function at the potential expense of global behavior—it can lead to purposefully ignoring information sources. Ergodicity insists on global coverage, anticipating spatially distributed uncertainties.

Ergodic control has formal properties (*e.g.*, asymptotic stability with respect to reference distributions [**140**]) that facilitate analysis. Metrics on ergodicity provide a *principle of motion* [**142**, **143**], similar to energy minimization and error minimization, and can be used to *synthesize* automation exploration for learning. As in [**144**–**146**], one can calculate a controller that optimizes the ergodic metric such that the trajectory of the robot is ergodic with respect to a target distribution. In this work, we use a receding-horizon controller based on minimizing a KL-ergodic measure detailed in [**144**] to synthesize future controls. The controller optimization minimizes the KL divergence $D_{KL}(h\|d)$, where h is a target distribution and d is the time-average statistics of the robot. The time-average statistics d of the robot are defined as

$$d(s|x(t)) = \frac{1}{T_r}\int_{t_i-T_r}^{t_i+T} \frac{1}{\eta}\psi(s|x(t))dt \tag{2.20}$$

where $\psi(s|x(t)) = \exp(-\frac{1}{2}\|s-x(t)\|^2_{\Sigma^{-1}})$, η is a normalization factor, s are states drawn from the reachable workspace, T is the planning horizon, and Σ specifies the width of the state coverage Gaussian.

There are many ways to specify a target spatial distribution h—one of which will be discussed further in Chapter 6. The key idea is that if we can specify a target distribution such that it represents the distribution of desired data in the test environment, an ergodic control method can be used to collect data to match the target distribution. Fig. 2–5 provides a visualization of a robot trajectory generated for an example target distribution with three

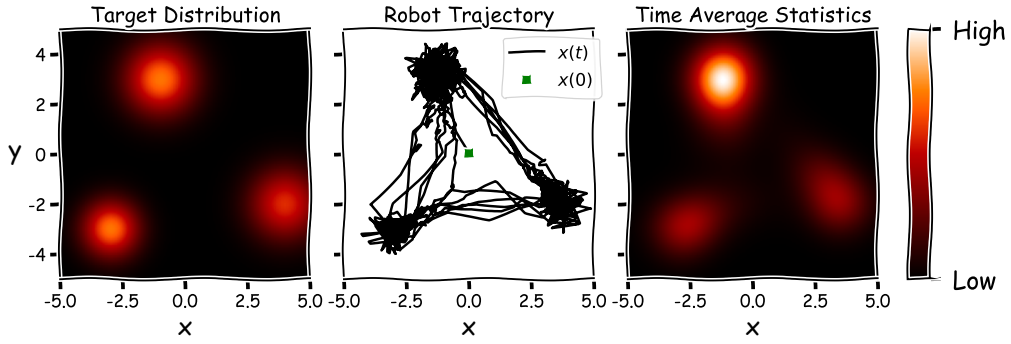

Figure 2–5 Ergodic Control Example: Ergodic control plans a future trajectory to minimize the difference between the target distribution (left) and the time average statistics (right) of the robot trajectory (middle). Regions of high interest are shown in white, while regions of low interest are shown in black.

regions of interest as well as the time average statistics of the robot trajectory. By inspection, the robot trajectory spends time collecting data in all three regions of interest.

2.5 Simulations and Benchmarking

Simulated benchmarks (*e.g.*, motor control problems and Atari games) are used extensively in RL. The benefits of these benchmarks are that they are safe, low-cost, repeatable, and (theoretically) able to run faster than in real time. Furthermore, simulations allow you to re-run a specific initial condition many times and observe how adjusting hyperparameters changes the learning process, while in physical environments, you can never run the exact same test twice. Although simulated benchmarks provide a common test platform for state-of-the-art (SOTA) methods, an algorithm showing *good performance* on simulated benchmarks often implies a (false sense of) general applicability to physical scenarios. The community is aware of the shortcomings of simple simulations and is working to build out more complex benchmarks [**46**, **147**–**150**]. But, phenomena like friction, slip, and contact are notoriously difficult to simulate, so it is not surprising that learning methods often take

advantage of the flaws specific to the simulation instead of learning something generalizable to other simulations and hardware testing.

There also exists a vast literature on sim-to-real transfer learning techniques that seek to bypass the challenges of real-world data collection [**40**]. Under this paradigm, simulators are leveraged for offline training, and real-world experience is used for online fine-tuning of robot policies [**44**]. While sim-to-real transfer techniques enable access to massively parallelized agent experience, they rely on access to high-fidelity physics simulators [**151**,**152**]. Moreover, simulators often incur expensive computational costs—especially when the task involves complex physical phenomena (*e.g.*, granular media and anisotropic frictional interfaces). Nonetheless, even the most sophisticated simulators fail to capture the richness of real-world physics beyond the most trivial environments. As a result, sim-to-real transfer agents still experience a degradation in policy performance referred to as the *reality gap* [**153**].

Just as some phenomena are difficult to simulate, many physical objects are also difficult to simulate. For example, imagine building a simulation to allow a robot to learn the difference between *real rocks* and *movie rocks*.[6] In simulation, the main feature we could manipulate would likely be mass, but this would disregard many of the complex and important features that distinguish *real rocks* from *movie rocks*. *Movie rocks* have different surface textures, compliance (*i.e.*, squishiness), and mass. Although we may visually be tricked into mistaking a "movie rock" for a *real rock*, the illusion ends as soon as we touch the object. And yet, encoding such a complex object into a simulation is difficult to imagine. By testing in simplified scenarios, are we limiting the learning representations our ML methods are able to generate?

[6] Movie rocks look like rocks but are actually foam.

While researchers have started to recognize the need for hardware learning benchmarks, they remain largely under-adopted by the field due to their reliance on bespoke software packages. Nonetheless, some researchers have developed hardware learning benchmarks for popular commercial robots [**154**,**155**], while others have developed remotely-accessible hardware platforms for the research community to deploy code onto remotely [**156**, **157**]. The downside to many of these approaches is that they rely on expensive robot hardware. Even encouraging recent work on the lower-cost end (*e.g.*, [**158**, **159**]) still cost upwards of $3.5k in component costs, so there is still a need for standardized, low-cost, open-source, hardware learning benchmarks in RL.

Since completely closing the *reality gap* is prohibitively challenging, learning algorithms capable of robustly navigating the complexities of real-world environments are essential. There is a pressing need for the development of hardware robotic benchmarks to evaluate and improve learning algorithms under realistic conditions. In this work, we present algorithms and approaches that enable embodied agents to learn from scratch in real-world scenarios.

Chapter 3

Hybrid Control for Combining Model-Based and Model-Free RL

In this chapter, we develop an approach to improve the capabilities of robotic systems by combining model-based and model-free Reinforcement Learning (RL). Learned predictive models provide an understanding of the task and the dynamics, while model-free (*i.e.*, experience-based) policy mappings encode favorable actions. Using hybrid control theory as the foundation for our approach, we derive a controller that optimally uses model-based actions when the policy is uncertain and allows the algorithm to fall back on the model-free policy when there is high confidence that the actions will result in a favorable outcome. Our approach, which we call hybrid learning, efficiently learns motor skills and improves the performance of predictive models and model-free policies. Moreover, our approach enables policies (both model-based and model-free) to be updated using any off-policy RL method.

In the following sections, we derive a deterministic hybrid learning method by optimally switching between learning modalities. Then, we adapt our method to a stochastic variation that relaxes some of the key assumptions in the original derivation. Finally, both variations are tested on a variety of robot control benchmark tasks in simulation as well as a hardware task. The results show that our method is capable of improving the performance and sample-efficiency of learning motor skills in a variety of experimental domains.

3.1 Hybrid Learning Approach

In this work, we build on the framework of Markov Decision Processes (MDPs) and aspects of hybrid control theory. Robot reinforcement learning problems are often formulated as MDPs. The MDP formulation assumes the Markov property—that the result of an action taken in a given state only depends on the current state and does not depend on the prior history. The Markov property applies to both model-based and model-free methods, which are used in this chapter. Prior work combining model-free and model-based methods often relies on hand-tuned objectives or ultimately solves two different objectives for the different learning methods. Instead, we introduce hybrid control theory as a structured approach to combining multiple control strategies with a single objective function.

Hybrid control theory for mode scheduling

In mode scheduling problems, the goal is to maximize a reward function through the synthesis of two (or more) control strategies. A common example of mode switching is when a vertical takeoff and landing vehicle switches from flight mode to landing mode. In hybrid control theory, these control strategies are often called modes. Thus, mode switching can also be called policy switching [**160**, **161**]. We can use hybrid control theory to determine the optimal time to switch from one control policy to another. Most mode scheduling problems are formulated in continuous time and are subject to dynamics of the form

$$\dot{s}(t) = f(s(t), a(t)) = g(s(t)) + h(s(t))a(t), \tag{3.1}$$

where $f(s, a) : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function. The state transition function is comprised of the free dynamics $g(s) : \mathcal{S} \to \mathcal{S}$, the control map $h(s) : \mathcal{S} \to \mathcal{S} \times \mathcal{A}$, and

an action $a(t)$. This action is assumed to be generated from some default behavior. This formulation also encompasses nonlinear state transition functions.

The continuous time dynamics function requires us to reformulate the objective from (2.1). First, we introduce an objective as a deterministic function of the state and control:

$$\arg\max_{\tau,\lambda} \mathcal{J} = \int_{t=0}^{t_H} r(s(t))dt \qquad \text{subject to } \dot{s}(t) = f(s(t), a(t)),\ s(0) = s_0 \tag{3.2}$$

where

$$a(t) = \begin{cases} \hat{a}(t), & \text{if } t \in [\tau, \tau + \lambda] \\ a_{\text{def}}(t) & \text{otherwise} \end{cases}, \tag{3.3}$$

t_H is the time horizon (in seconds) and $s(t)$ is generated from some initial condition $s(0)$ using a model (3.1) and an action sequence (3.3). The goal in hybrid control theory is to find the optimal time τ to switch from some default set of actions a_{def} to another set of actions $\hat{a}$ for application duration λ to best improves the performance of the objective in (3.2) subject to the dynamics (3.1).

Connection to Markov Decision Processes

In Section 2.1.2, we introduced MDPs and described how model-based and model-free methods follow the MDP formulation. Although both methods assume an underlying MDP, only the model-based method learns this transition probability. The transition probability is a key feature linking the MDP formulation to hybrid control theory. Most mode scheduling problems are constrained by a dynamics function that governs the transition from a current state s_t to a next state s_{t+1} given a current action a_t. Thus, the MDP transition probability and the mode scheduling dynamics function have the same form. We use the model-based prediction capability to develop a hybrid control approach to determine the optimal time to switch from the action specified by model-free policy to an alternate model-based action.

3.1.1 Deterministic Method

Consider a continuous time MDP formulation of the objective in (3.2) and the dynamics in (3.1), where the dynamics f and reward function r are learned using regression methods (*e.g.*, neural network least squares or Gaussian processes), and the model-free policy π is learned through a model-free approach (*e.g.*, policy gradient [**162**]). We assume the learned policy has the form $\pi(a|s) = \mathcal{N}(\mu(s), \Sigma(s))$, where $\mathcal{N}$ is a normal distribution and $\mu(s)$, $\Sigma(s)$ are the mean and variance of the policy as a function of state. For now, we ignore uncertainty[1] and define the default action in (3.3) as the mean of the policy π, so $a_{\text{def}}(t) = \mu(s(t))$. To determine the form of $\hat{a}$, let us first calculate how sensitive the objective in (3.2) is at any τ to switching from $\mu(s) \to \hat{a}$ for an infinitely small λ.[2]

Lemma 3.1 (Mode Insertion Gradient) *Assume (f, r, μ) are differentiable and continuous in time. The sensitivity of the objective in (3.2) with respect to the duration time λ of switching from $\mu(s)$ to $\hat{a}$ at any time $\tau \in [0, t_H]$ is defined as*

$$\frac{\partial}{\partial \lambda} \mathcal{J}(\tau) = \rho(\tau)^\top (f_2 - f_1)\Big|_\tau \tag{3.4}$$

where $f_1 = f(s(t), \mu(s(t)))$, $f_2 = f(s(t), \hat{a}(t))$, and $\rho(t) \in \mathcal{S}$. The adjoint variable ρ is the solution to the differential equation

$$\dot{\rho}(t) = -\frac{\partial r}{\partial s} - \left(\frac{\partial f}{\partial s} + \frac{\partial \mu}{\partial s}^\top \frac{\partial f}{\partial a}\right)^\top \rho(t) \tag{3.5}$$

*with terminal condition $\rho(t_H) = \mathbf{0}$. The derivative of the objective in (3.2) with respect to time duration λ is also known as the mode insertion gradient [**160, 161**].*

---

[1] We add uncertainty in the stochastic derivation of our hybrid learning approach.

[2] We avoid instabilities from switching control strategies by finding the best action for all $\tau \in [0, t_H]$ instead of searching for a particular switching time.

Proof. First, we define the trajectory

$$s(t_H) = s(0) + \int_0^{\tau} f(s(t), \mu(s(t)))\, dt + \int_{\tau}^{\tau+\lambda} f(s(t), \hat{a}(t))\, dt + \int_{\tau+\lambda}^{t_H} f(s(t), \mu(s(t)))\, dt \quad (3.6)$$

generated from $a(t) = \begin{cases} \hat{a}(t), & \text{if } t \in [\tau, \tau+\lambda] \\ a_{\text{def}}(t) & \text{otherwise} \end{cases}$ where $a_{\text{def}}(t) = \mu(s(t))$. Next, we take the derivative of (3.2) with respect to the time duration λ to get the expression

$$\frac{\partial}{\partial \lambda} \mathcal{J} = \int_{\tau+\lambda}^{t_H} \frac{\partial r}{\partial s}^{\top} \frac{\partial s}{\partial \lambda}\, dt. \quad (3.7)$$

Using (3.6), we define the derivative of the state s with respect to the time duration λ as

$$\frac{\partial s(t)}{\partial \lambda} = f_2 - f_1 + \int_{\tau+\lambda}^{t} \left(\frac{\partial f}{\partial s} + \frac{\partial \mu}{\partial s}^{\top} \frac{\partial f}{\partial a} \right)^{\top} \frac{\partial s(\sigma)}{\partial \lambda}\, d\sigma \quad (3.8)$$

where $f_1 = f(s(t), \mu(s(t)))$ and $f_2 = f(s(t), \hat{a}(t))$ are boundary terms from applying Leibniz's rule and σ is a placeholder for time under the integrand. The derivative in (3.8) is a linear convolution with initial condition $\frac{\partial s(\tau)}{\partial \lambda} = f_2 - f_1$, so we can rewrite (3.8) using a state-transition matrix as

$$\frac{\partial s(t)}{\partial \lambda} = \Phi(t, \tau)(f_2 - f_1) \quad (3.9)$$

where

$$\Phi(t, \tau) = \exp\left(\left(\frac{\partial f}{\partial s} + \frac{\partial \mu}{\partial s}^{\top} \frac{\partial f}{\partial a} \right)^{\top} (t - \tau) \right). \quad (3.10)$$

Plugging (3.9) into (3.7) and pulling the initial condition term out of the integrand yields

$$\frac{\partial}{\partial \lambda} \mathcal{J}(\tau) = \lim_{\lambda \to 0} \int_{\tau+\lambda}^{t_H} \frac{\partial r}{\partial s}^{\top} \Phi(t, \tau)\, dt\, (f_2 - f_1). \quad (3.11)$$

Taking the limit of (3.11) as $\lambda \to 0$ gives the instantaneous sensitivity from switching from $\mu \to \hat{a}$ at any time τ. We define this sensitivity term as the adjoint variable

$$\rho(\tau)^{\top} = \int_{\tau}^{t_H} \frac{\partial r}{\partial s}^{\top} \Phi(t, \tau)\, dt. \quad (3.12)$$

Plugging the adjoint variable back into (3.11) gives the mode insertion gradient

$$\frac{\partial}{\partial \lambda} \mathcal{J}(\tau) = \rho(\tau)^\top \left(f_2 - f_1 \right), \tag{3.13}$$

where the adjoint can be rewritten as the differential equation

$$\dot{\rho}(t) = -\frac{\partial r}{\partial s} - \left(\frac{\partial f}{\partial s} + \frac{\partial \mu}{\partial s}^\top \frac{\partial f}{\partial a} \right)^\top \rho(t) \tag{3.14}$$

with terminal condition $\rho(t_H) = \mathbf{0}$. □

Lemma 3.1 gives us the proof and definition of the mode insertion gradient (3.4). The mode insertion gradient tells us the change in the objective function when switching from the default policy behavior μ to some other arbitrarily defined control $\hat{a}$ for a small time duration λ. The mode insertion gradient can show how an arbitrary action changes task performance relative to the learned policy. In this chapter, we additionally use the mode insertion gradient as a method for obtaining the best action the robot can take given the learned predictive model of the dynamics and the task rewards.

We can take a direct approach by asking the following question: Given a suboptimal policy π, what is the best action the robot can take to maximize the objective in (3.2), at any time $t \in [0, t_H]$, subject to the uncertainty (or certainty) of the policy defined by $\Sigma(s)$? We approach this new sub-problem by specifying the auxiliary optimization problem

$$a^\star(t) = \underset{\hat{a}(t)\, \forall\, t \in [0, t_H]}{\arg\max} \int_0^{t_H} \underbrace{\frac{\partial}{\partial \lambda} \mathcal{J}(t)}_{\text{mode insertion gradient}} + \underbrace{\log \pi\Big(\hat{a}(t) | s(t)\Big)}_{\text{deviation penalty}} \, dt. \tag{3.15}$$

This new problem maximizes the mode insertion gradient by finding the action $a^\star$ that results in the greatest change in the objective. The problem also penalizes the action $a^\star$ for deviating from the policy when there is high confidence in the policy from prior experience.

Theorem 3.1 (Optimal Action) *Assuming (f, r, π) are all continuous and differentiable in (s, a, t), the best possible action to improve the performance of (3.2) and is a solution to (3.15) for any time $t \in [0, t_H]$ is*

$$a^\star(t) = \Sigma(s(t))h(s(t))^\top \rho(t) + \mu(s(t)) \tag{3.16}$$

where the adjoint $\rho(t)$ is defined in (3.5) and $h(s) : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ is the affine mapping from actions to the dynamics.

Proof. The mode insertion gradient can be rewritten (3.4) as

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial \lambda} &= \rho^\top \Big(f(s, \hat{a}) - f(s, \mu(s)) \Big) \\ &= \rho^\top \Big(g(s) + h(s)\hat{a} - g(s) - h(s)\mu(s) \Big) \\ &= \rho^\top h(s) \Big(\hat{a} - \mu(s) \Big) \end{aligned} \tag{3.17}$$

where $f(s, a) = g(s) + h(s)a$, and we drop the dependency on time for clarity. Plugging the mode insertion gradient into (3.15), taking the derivative with respect to the point-wise $\hat{a}$, and setting the equation equal to zero gives

$$\rho^\top h(s) \left(\hat{a} - \mu(s) \right) - \Sigma(s)^{-1} \left(\hat{a} - \mu(s) \right) = 0. \tag{3.18}$$

Solving for $\hat{a}$ gives the best actions $a^\star$

$$a^\star(t) = \Sigma(s(t))h(s(t))^\top \rho(t) + \mu(s(t)), \tag{3.19}$$

which is the action that maximizes the mode insertion gradient subject to the certainty of the policy π for all $t \in [0, t_H]$. □

The proof in Theorem 3.1 provides the best action a robotic system can take given a default model-free policy. Each action generated uses the sensitivity of changing the objective

based on the predictive model's behavior while relying on the model-free policy to regulate when the model information will be useful. We convert the result in Theorem 3.1 into our first algorithm, a deterministic approach to hybrid learning (see Algorithm 3–1).

Algorithm 3–1 Hybrid Learning (Deterministic)

Randomly initialize continuous differentiable models $f(s,a)$, $r(s,a)$ with parameters ψ and policy $\pi(a|s)$ with parameter θ. Initialize memory buffer $\mathcal{D}$, prediction horizon parameter t_H, exploration noise ε.
while task not done **do**
  Reset environment and exploration noise ε
  for $i = 1, \ldots, T-1$ **do**
    Observe state $s(t_i)$
    for $\tau_i \in [t_i, \ldots, t_i + t_H]$ **do** ▷ Forward simulate dynamics and rewards
      $s(\tau_{i+1}) = s(\tau_i) + f(s(\tau_i), \mu(s(\tau_i)))dt$
      $r(\tau_i) = r(s(\tau_i), \mu(s(\tau_i)))$
    end for
    $\rho(t_i + t_H) = \mathbf{0}$
    for $\tau_i \in [t_H + t_i, \ldots, t_i]$ **do** ▷ Backwards integrate
      $$\dot{\rho}(t) = -\frac{\partial r}{\partial s} - \left(\frac{\partial f}{\partial s} + \frac{\partial \mu}{\partial s}^\top \frac{\partial f}{\partial a}\right)^\top \rho(t)$$
      $\rho(\tau_{i-1}) = \rho(\tau_i) - \dot{\rho}(\tau_i)dt$
    end for
    $a^\star(t_i) = \Sigma(s(t_i))h(s(t_i))^\top \rho(t_i) + \mu(s(t_i)) + \varepsilon(t)$
    Apply to robot and observe reward r_t
    Append data to buffer $\mathcal{D} \leftarrow \mathcal{D} + \{s(t_i), a^*(t_i), r_t\}$
  end for
  Update f, r by sampling N data points from $\mathcal{D}$ using any regression method ▷ Model [3]
  Update π with any experience-based method ▷ Policy [3]
end while

[3] Updates can also be done inside the for loop after enough samples have been collected in $\mathcal{D}$

With this approach, we are able to numerically determine the contribution of the learned predictive models towards improving the task. We show (3.16) provides the best possible action given the current belief of the dynamics f and the task reward r. Next, we use the control Hamiltonian to show we can find a local optima of the objective in (3.16).

Corollary 3.1 (Control Hamiltonian) *Assuming* $\frac{\partial}{\partial a}\mathcal{H} \neq 0$ *where* $\mathcal{H} = r(s) + \log \pi(a|s) + \rho^\top f(s,a)$ *is the control Hamiltonian for the objective in (3.2), then* $\frac{\partial}{\partial \lambda}\mathcal{J} = \|h(s)^\top \rho\|_{\Sigma(s)} > 0$ *and is zero when the policy satisfies the control Hamiltonian condition* $\frac{\partial}{\partial a}\mathcal{H} = 0$.

Proof. Inserting (3.16) into (3.4) yields

$$\begin{aligned}\frac{\partial \mathcal{J}}{\partial \lambda} &= \rho^\top \Big(g(s) + h(s) \left(\Sigma(s) h(s)^\top \rho + \mu(s) \right) - g(s) - h(s)\mu(s) \Big) \\ &= \rho^\top h(s) \Sigma(s) h(s)^\top \rho \\ &= \|h(s)^\top \rho\|_{\Sigma(s)} > 0. \end{aligned} \tag{3.20}$$

From Pontryagin's maximum principle, a solution is a local optimum of the objective function if it satisfies the following

$$\frac{\partial}{\partial a}\mathcal{H} = -\Sigma(s)^{-1} \left(a - \mu(s) \right) + h(s)^\top \rho = 0 \tag{3.21}$$

when $a = \Sigma(s) h(s)^\top \rho + \mu(s)$ or $\rho = 0$. Therefore, if the policy π is a solution, then the adjoint $\rho = 0$, and π must be a solution to the optimal control problem (3.2). □

Corollary 3.1 tells us the action defined in (3.16) generates the best action to improve the performance of the robot given valid learned models. Corollary 3.1 also states that if the policy is already a solution, then our approach for hybrid learning does not modify the known solution and simply returns the policy's action.

The preceding proofs have a strict requirement of continuity and differentiability of the learned models and the policy. It is not always possible to satisfy these constraints, and often learned models have noisy derivatives. Therefore, our goal is to reformulate the objective from (3.2) into an equivalent problem that can be solved without these strict assumptions. In the next section, we introduce an approach to reformulate the objective.

3.1.2 Stochastic Method

For the stochastic method, we relax the continuity, differentiability, and continuous-time restrictions specified in the deterministic objective in (3.2) by first restating the objective as an expectation

$$\max \mathop{\mathbb{E}}_{v\sim\pi(\cdot|s)}\Big[\mathcal{J}(v)\Big], \text{ where } \mathcal{J}(v) = \sum_{t=0}^{T-1} r(s_t) \text{ s.t. } s_{t+1} = f(s_t, v_t) \tag{3.22}$$

f is the transition dynamics and v=$[v_0, \ldots, v_{H-1}]$ is a sequence of H randomly generated actions from the model-free policy π. Rather than trying to find the best time τ and discrete duration λ, we approach the problem from a hybrid information theoretic view. In this framework, we want to find the best actions to augment π and improve the objective.

We find the best augmented action by defining two distributions $\mathbb{P}$ and $\mathbb{Q}$. Distribution $\mathbb{P}$ is the uncontrolled system response distribution,[4] and distribution $\mathbb{Q}$ is the open-loop control distribution. Distributions $\mathbb{P}$ and $\mathbb{Q}$ are described by probability density functions

$$p(v) = \prod_{t=0}^{T-1} \pi\left(v_t|s_t\right) \quad \text{and} \quad q(v|a) = \prod_{t=0}^{T-1} \frac{\exp\left(-\frac{1}{2}(v_t - a_t)^\top \Sigma(s_t)^{-1}(v_t - a_t)\right)}{\sqrt{(2\pi)^m|\Sigma(s_t)|}} \tag{3.23}$$

where $q(v|a)$ uses the same variance $\Sigma(s)$ as the policy $\pi(a|s) = \mathcal{N}(\mu(s), \Sigma(s))$. The uncontrolled distribution $\mathbb{P}$ represents the default predicted behavior of the robotic system under the learned policy π. The open-loop control distribution $\mathbb{Q}$ allows us to define a probability of an augmented action, but more importantly, distribution $\mathbb{Q}$ provides a free variable, which we will use to optimize the learned models.

[4] We refer to uncontrolled as the un-augmented control response of the robotic agent subject to a stochastic model-free policy π.

Following the work in [**2**], we use Jensen's inequality and importance sampling on the free-energy definition [**163**] of the control system using distribution $\mathbb{Q}$ to get the relationship

$$\mathcal{F}(v) = -\frac{1}{\gamma}\log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(\gamma\mathcal{J}(v)\right)\right]\right) \leq -\frac{1}{\gamma}\mathbb{E}_{\mathbb{Q}}\left[\log\left(\frac{p(v)}{q(v|a)}\exp\left(\gamma\mathcal{J}(v)\right)\right)\right] \tag{3.24}$$

where $\gamma \in \mathbb{R}^+$ here is the inverse temperature parameter. In (3.24), if

$$\frac{p(v)}{q(v|a)} \propto 1/\exp\left(\gamma\mathcal{J}(v)\right), \tag{3.25}$$

then the inequality becomes a constant. Further reducing the free-energy gives

$$\mathcal{F}(v) \leq -\mathbb{E}_{\mathbb{Q}}\left[\mathcal{J}(v) - \frac{1}{\gamma}\log\left(\frac{p(v)}{q(v|a)}\right)\right], \tag{3.26}$$

which is the optimal control problem we want to solve plus a bounding term. The bounding term keeps the augmented actions close to the policy. By making the bound a constant, we can use the free-energy formulation to solve the hybrid control problem indirectly.

Now, we can define an optimal distribution $\mathbb{Q}^\star$ as a density function

$$q^\star(v) = \frac{1}{\eta}\exp\left(\gamma\mathcal{J}(v)\right)p(v), \tag{3.27}$$

where $\eta = \int_\Omega \exp\left(\gamma\mathcal{J}(v)\right)p(v)dv$ and Ω is the sample space.[5] We can use the ratio

$$\frac{p(v)}{q^\star(v)} \propto 1/\exp\left(\gamma\mathcal{J}(v)\right) \tag{3.28}$$

to make the free-energy a constant. However, we cannot directly sample from $\mathbb{Q}^\star$. We want to generate a separate set of actions a_t defined in $q(v|a)$ to augment the policy π given the learned model f, so our goal is to push $q(v|a)$ towards $q^\star(v)$. As done in [**2**, **164**], this goal corresponds to the optimization

$$a^\star = \arg\min_a D_{\mathrm{KL}}\left(\mathbb{Q}^\star|\mathbb{Q}\right). \tag{3.29}$$

[5] Similar to the mode insertion gradient from the deterministic approach, the optimal density function can be used to determine how much the policy π needs to be augmented to maximize the objective.

The optimization minimizes the Kullback-Leibler (KL) divergence of the optimal distribution $\mathbb{Q}^\star$ and the augmented open-loop distribution $\mathbb{Q}$.

Returning to our original problem of how to find the best actions to augmented π and improve the objective in (3.22), we now we want to construct a distribution to augment the policy distribution $p(v)$ and improve the objective.

Theorem 3.2 *The recursive, sample-based solution to (3.29) is*

$$a_t^\star = a_t + \sum_k \omega(v_t^k)\delta a_t^k \quad \textit{where} \quad \omega(v) = \frac{\exp\left(\gamma \mathcal{J}(v)\right) p(v)}{\sum_k \exp\left(\gamma \mathcal{J}(v)\right) p(v)}, \tag{3.30}$$

k denotes the sample index and $v_t = a_t + \delta a_t$.

Proof. Expanding the objective in (3.29), we can show

$$\begin{aligned}
a^\star &= \arg\min_a \mathbb{E}_{\mathbb{Q}^\star}\left[\log\left(\frac{q^\star(v)}{q(v|a)}\right)\right] \\
&= \arg\min_a \int_\Omega q^\star(v) \log\left(\frac{q^\star(v)}{p(v)}\frac{p(v)}{q(v|a)}\right) dv \\
&= \arg\min_a \left(\int_\Omega q^\star(v) \log\left(\frac{q^\star(v)}{p(v)}\right) dv - \int_\Omega q^\star(v) \log\left(\frac{q(v|a)}{p(v)}\right) dv\right) \\
&= \arg\max_a \int_\Omega q^\star(v) \log\left(\frac{q(v|a)}{p(v)}\right) dv.
\end{aligned} \tag{3.31}$$

Defining the policy $\pi(v_t|s_t) = \mathcal{N}\left(\mu(s_t), \Sigma(s_t)\right)$ as normally distributed, we can show

$$\begin{aligned}
\frac{q(v|a)}{p(v)} &\propto \exp\left(\sum_t \Big(-\frac{1}{2}(v_t - a_t)^\top \Sigma^{-1}(v_t - a_t) + \frac{1}{2}(v_t - \mu(s_t))^\top \Sigma^{-1}(v_t - \mu(s_t))\Big)\right) \\
&= \exp\left(\sum_t \Big(-\frac{1}{2}a_t^\top \Sigma^{-1} a_t + a_t^\top \Sigma^{-1} v_t + \frac{1}{2}\mu(s_t)^\top \Sigma^{-1}(\mu(s_t) - 2v_t)\Big)\right)
\end{aligned}$$

where $\Sigma = \Sigma(s)$ is used for clarity.

Plugging this expression into (3.31) gives

$$a^\star = \arg\max_a \left(\sum_t \left(\frac{1}{2} a_t^\top \Sigma^{-1} a_t + a_t^\top \int_\Omega q^\star(v) \Sigma^{-1} v_t dv + \frac{1}{2} \mu(s_t)^\top \int_\Omega q^\star(v) \Sigma^{-1} (\mu(s_t) - 2v_t) dv \right) \right). \tag{3.32}$$

We can solve (3.32) for a_t at each time by setting the derivative with respect to a_t equal to zero. Thus, we get the optimal solution

$$a_t^\star = \int_\Omega q^\star(v) v_t dv. \tag{3.33}$$

The expression $q^\star(v) \propto \exp\left(\gamma \mathcal{J}(v)\right) p(v)$ allows us to rewrite (3.33) as

$$\begin{aligned} a_t^\star &= \int_\Omega q^\star(v) v_t dv \\ &= \int_\Omega \frac{1}{\eta} \exp\left(\gamma \mathcal{J}(v)\right) p(v) v_t dv \\ &= \mathbb{E}_{\mathbb{P}} \left[\frac{1}{\eta} \exp\left(\gamma \mathcal{J}(v)\right) v_t \right], \end{aligned} \tag{3.34}$$

where $\eta = \int_\Omega \exp\left(\gamma \mathcal{J}(v)\right) p(v) dv$. Using the change of variable $v_t = a_t + \delta a_t$, we get the recursive, sample-based solution

$$a_t^\star = a_t + \sum_k \omega(v_t^k) \delta a_t^k \quad \text{where} \quad \omega(v) = \frac{\exp\left(\gamma \mathcal{J}(v)\right) p(v)}{\sum_k \exp\left(\gamma \mathcal{J}(v)\right) p(v)}. \tag{3.35}$$

□

With the stochastic formulation, we can generate samples from the stochastic policy and evaluate its utility based on the current belief of the dynamics and the reward function. Because samples directly depend on the likelihood of the policy, any actions deviating too far from the policy will be penalized proportional to the confidence of the policy. The inverse is also true—when the policy has low confidence (high variance), the sample span will increase, and the model-based information will have higher weight. Additionally, we do

not have to place continuity and differentiability constraints on the learned models and can utilize arbitrarily complex models in this algorithm. We outline the stochastic algorithm for hybrid learning Algorithm 3–2.

Algorithm 3–2 Hybrid Learning (Stochastic)

Randomly initialize continuous differentiable models $f(s,a)$, $r(s,a)$ with parameters ψ and policy $\pi(a|s)$ with parameter θ. Initialize memory buffer $\mathcal{D}$, prediction horizon parameter H, sample size parameter K, inverse temperature parameter γ.
while task not done **do**
    Reset environment
    for $t = 1, \ldots, T-1$ **do**
        Observe state s_t
        for $k \in \{0, \ldots, K-1\}$ **do** ▷ Forward simulate
            for $\tau \in \{0, \ldots, H-1\}$ **do** ▷ Predict states and rewards
                $v_\tau^k \sim \pi(\cdot|s_\tau^k)$
                $s_{\tau+1}^k, r_\tau^k = f(s_\tau^k, v_\tau^k), r(s_\tau^k, v_\tau^k)$
                $j_\tau^k = r_\tau^k$
            end for
        end for
        for $\tau \in \{0, \ldots, T-1\}$ **do** ▷ Update Actions
            $\mathcal{J}(v_\tau^k) \leftarrow \sum_{t=\tau}^{T-1} j_t^k$
            $\delta a_\tau^k \leftarrow v_\tau^k - a_\tau$
            $\omega(v_\tau^k) \leftarrow \dfrac{\exp\left(\gamma \mathcal{J}(v_\tau^k)\right) p(v_\tau^k)}{\sum_k \exp\left(\gamma \mathcal{J}(v_\tau^k)\right) p(v_\tau^k)}$
            $a_\tau \leftarrow a_\tau + \sum_{k=0}^{K-1} \omega(v_\tau^k) \delta a_\tau^k$
        end for
        Apply a_0 to robot and observe reward r_t
        Append data to buffer $\mathcal{D} \leftarrow \mathcal{D} + \{s_t, a_0, r_t\}$
    end for
    Update f, r by sampling N data points from $\mathcal{D}$ using any regression method ▷ Model [6]
    Update π with any experience-based method ▷ Policy [6]
end while

[6] Updates can also be done inside the for loop after enough samples have been collected in $\mathcal{D}$

3.2 Experiments

This section compares the previously described methodology to state of the art model-based and model-free methods. First, we evaluate our deterministic and stochastic variations on a range of simulated benchmark control tasks from OpenAI Gym [**12**]. Then, we extend the evaluation of our stochastic method to the real-world with a robot arm experiment. See Appendix A for implementation details. Unless otherwise stated, the solid curves in the following figures correspond to the mean, and the shaded regions correspond to the standard deviation over the ten trials (with different random seeds).[7]

3.2.1 Simulated Benchmarks

We evaluate our approach on a subset of environments from the OpenAI Gym benchmarks [**12**]. Specifically we look at the cartpole swing-up environment, the Acrobot swing-up environment, the hopper environment, and the half-cheetah environment, which are described in more detail in Appendix A. We evaluate our proposed approach on common RL benchmarks to verify our measure for model and policy alignment. In particular, the goal is to show that our method 1) is more sample efficient, 2) obtains higher rewards than the comparison methods even though it is a model-based RL method, and 3) improves the performance of the individual model and policy that are being learned over time.

Deterministic Results. We compare our hybrid learning approach to model-based method in [**1**] and model-free method Soft Actor Critic (SAC) in [**3**]. While model-free methods have exploration naturally encoded in their formulation, the deterministic model-based and hybrid learning approaches require added exploration noise to induce exploring other regions

[7] For results and code please visit `https://github.com/MurpheyLab/HybridLearning`.

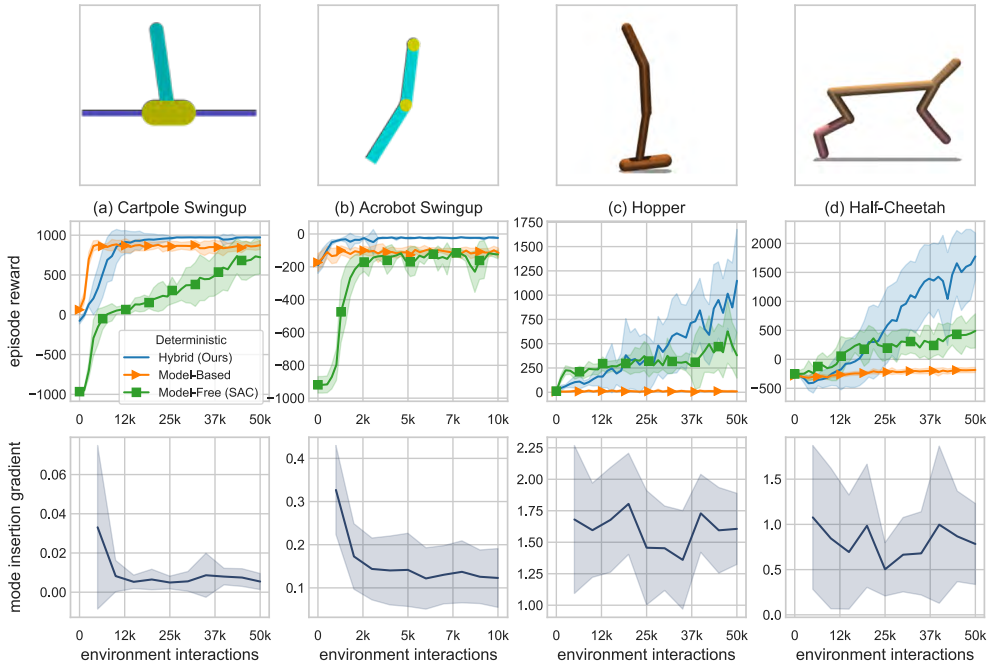

Figure 3–1 Performance curves of our deterministic hybrid learning algorithm. All methods use the same structured learning models. Our method is shown to improve the model-based benchmark results (due to the use of experience-based methods) while maintaining significant improvements on the number of interactions necessary with the environment to obtain those results. The mode insertion gradient is also shown for each exampler. The mode insertion gradient illustrates the model-policy agreement over time. Note: Markers are intended to distinguish between methods and do not represent all data points. Results are averaged over 10 random seeds.

of state-space. Fig. 3–1 summarizes our deterministic benchmark results. The top row shows snapshots of each environment. The middle row shows that our method outperforms both the model-based and model-free method for all four tested environments. Our hybrid learning approach uses the confidence bounds generated by the stochastic policy to infer when best to rely on the policy or predictive models. As a result, hybrid learning enables learning in unstructured environments comparable to model-free methods with the sample-efficiency of model-based learning approaches. For the swing-up tasks, the model-based method learns the task but converges to a less optimal policy than our hybrid approach. For the locomotion tasks, the model-based method suffers from its inability to model discontinuous dynamics.

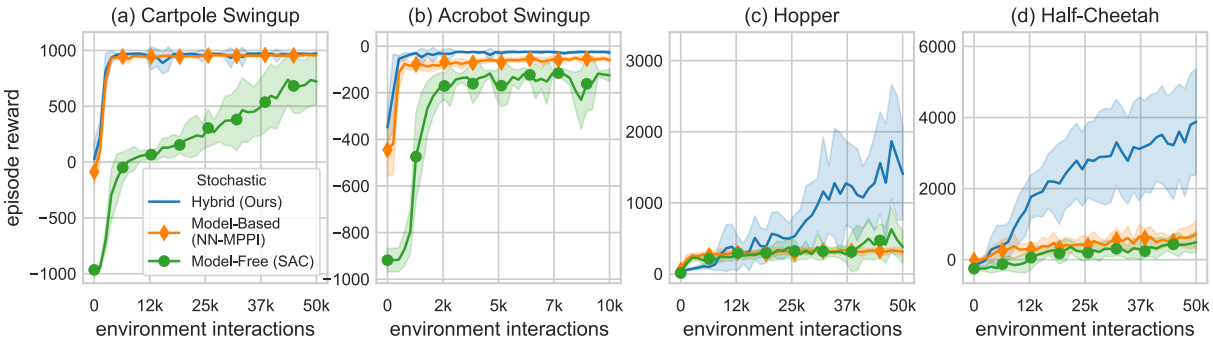

Figure 3–2 Performance curves of our stochastic hybrid learning algorithm. Test environments are shown in Fig. 3–1. Our approach improves both the sample-efficiency and the highest expected reward. Note: Markers are intended to distinguish between methods and do not represent all data points. Results are averaged over 10 random seeds.

The bottom row shows the mode insertion gradient for the hybrid learning method for each environment. The mode insertion gradient can be interpreted as a measure of agreement between the policy and the learned model. The swing-up examples show an eventual reduction in the mode insertion gradient. We interpret this as the convergence of the learned model and policy. The locomotion examples do not converge in the number of environment interactions observed. The lack of convergence indicates that the learning process has not yet stabilized, which is consistent with the episode rewards learning curves (middle row).

Stochastic Results. We next compare the stochastic formulation of hybrid learning to a stochastic neural-network model-based method, Neural-Network Model Predictive Path Integral Control (NN-MPPI) [**2**] and the same model-free controller as the deterministic formulation (SAC) [**3**]. The goal of these experiments is to show that this hybrid learning formulation still outperforms these state of the art learning techniques without imposing continuity and differentiability requirements. Fig. 3–2 summarizes our stochastic benchmark results. The model-free parameters are held constant from the deterministic results to remove any impact of hyperparameter tuning. Fig. 3–2 shows that for both swing-up tasks, the

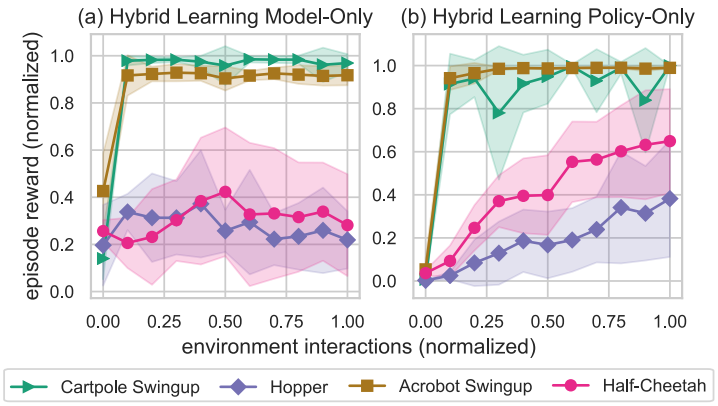

Figure 3–3 Comparison of model and policy evolution for stochastic hybrid learning. Episode rewards and environment interactions are normalized to allow comparisons between different environments. For the swing-up tasks, both the models and policies quickly learn the task. For the locomotion tasks, models also learn over the course of training, but the policies learn at a faster rate. These plots show that the hybrid learning approach improves the performance of both the model and the policy. Note: Markers are intended to distinguish between methods and do not represent all data points. Results are averaged over 10 random seeds.

stochastic formulation outperforms the model-free method but performs similarly to the improved model-based method. Even with the improved model-based method, the model-based method converges to a less optimal policy for the Acrobot than our hybrid approach. For the locomotion tasks, our stochastic formulation maintains the improved performance and sample-efficiency over the model-based method and the model-free method. Exploration is naturally encoded into the stochastic algorithm, which results in more stable learning when there is uncertainty in the task.

Although we no longer have a mode insertion gradient for the stochastic method, we can analyze the individual learned model and policy obtained from hybrid learning. In Fig. 3–3, both the environment interactions and episode rewards are normalized to allow comparisons between the different environments. For all environments, hybrid learning is shown to improve the learning capabilities of both the learned predictive model and the policy through the hybrid control approach. The policy is "filtered" through the learned

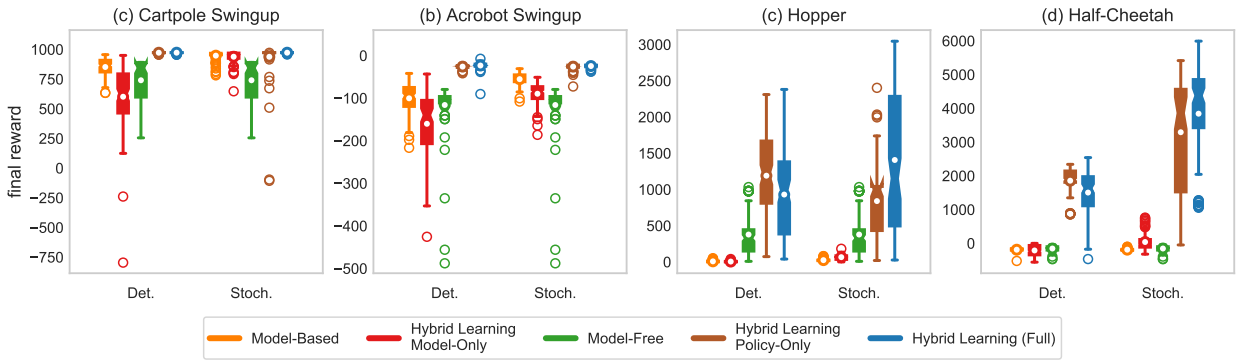

Figure 3–4 Comparison of final model-based and model-free policies for deterministic and stochastic methods. For the hybrid learning method, we show the final model-only and policy-only results separately as well as in combination. Each boxplot displays the results of 100 tests (10 seeds, 10 episodes each). The interquartile region (IQR) contains 50% of the data (from Q1 to Q3) and is shown as a filled box with notches at the median. The whiskers span [Q1 - 1.5×IQR, Q3 + 1.5×IQR]. Any data points falling outside the whiskers are designated as outliers and are shown as unfilled circles. The means are shown as white circles. For all methods, the hybrid policy-only and full hybrid learning outperform all other methods.

model and augmented, allowing the robotic system to be guided by both the prediction and experience. Thus, both the predictive model and the policy benefit, ultimately performing better as a standalone approach using hybrid learning. Fig. 3–3 shows that for different tasks, the model and policy learn at different rates, but both evolve over time with our hybrid learning approach.

Hybrid Learning Comparisons. We can compare the learned models and policies at 50,000 steps for both the deterministic and stochastic implementations. Fig. 3–4 shows 10 tests of each final model-based, model-free, and hybrid learning method. In addition to the full hybrid method, we show the individual model-based and model-free policies trained during hybrid learning as "Hybrid Learning Model-Only" and "Hybrid Learning Policy-Only" respectively. In addition to having greater performance with the full implementation of hybrid learning, our approach tends to push the performance of both the model and

policy. For the swing-up tasks, the model-based method outperformed the hybrid model only evaluation, but both the full hybrid learning and hybrid policy only evaluations outperform all other methods. These results indicate that the hybrid learning approach does not require model improvements once it has learned a "good enough" model.

For the locomotion tasks, the deterministic policy-only hybrid learning on average outperforms the deterministic full hybrid learning approach. For the stochastic method, the results are reversed with the full hybrid learning approach outperforming the policy-only approach. These results indicate that if the goal is to produce animations in simulation, the policy-only controller is likely sufficient to control the simulation. If the goal is to control real-world systems with uncertainty, maintaining the full hybrid controller allows continued benefit from the learned predictive model and model-free policy.

3.2.1.1 Related Work Benchmarking. Our hybrid learning approach is able to leverage both the model and the policy during both training and real time operation. This is a major difference between our approach and other RL methods which combine model-based and model-free approaches. Many approaches in related work only use the model to help the model-free policy learn better but do not have any use for the model once the model-free policy training is complete.

Methods that only the model-free policy to interact with the environment both during training and after training include Model-Based Value Expansion (MVE) [**4**], Stochastic Ensemble Value Expansion (STEVE) [**5**], and Model Based Policy Optimization (MBPO) [**6**]. For these methods, the model is learned offline using data accumulated by the model-free policy, but there is no model-based controller associated with the model. Instead, the policy is used to "imagine" future transitions in the model-based environment for short rollouts.

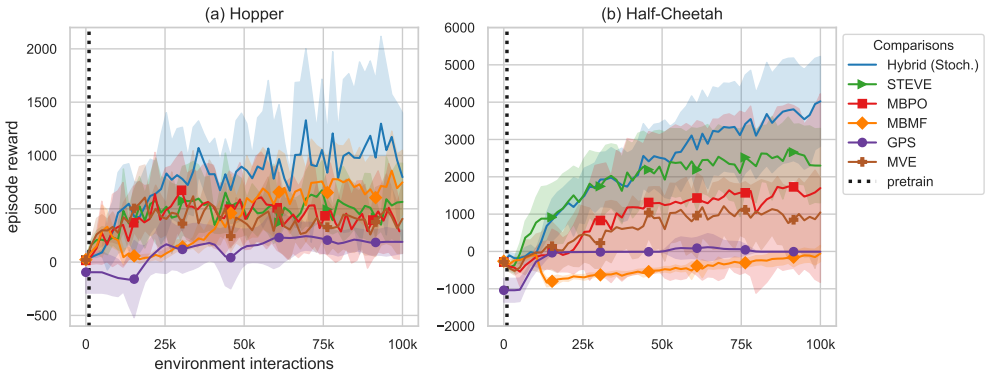

Figure 3–5 Comparison to related work combining model-based and model-free methods. Our approach improves both the sample-efficiency and the highest expected reward. Additional details and pairwise comparisons are included in Appendix A.2. Note: Markers are intended to distinguish between methods and do not represent all data points. Results are averaged over 10 random seeds.

These model rollouts are used during the update step to augment Q-learning (for MVE and STEVE) or the policy directly (for MBPO).

Guided Policy Search (GPS) [**7**] takes a different approach by using a model-based controller to interact with the environment. During each exploration iteration, GPS fits a dynamics model to the most recently collected data and uses this model to constrain the policy updates. In this algorithm, only the model-based controller interacts with the environment during training, and the goal is to train a policy which can be used without the model after training.

Model-based learning with model-free fine-tuning (MB-MF) [**8**] takes another approach. First, a model-based random shooting controller is used to learn a coarse-grained representation of the dynamics. Then, the learned dynamics are transferred to a model-free method via data aggregation. Finally, the model-free policy is fine-tuned through interactions with the environment. Both the model-based controller and the model-free policy interact with the environment, but they do so sequentially—during any given trial, only one of them plays a role. Although the model is no longer needed once the model-free policy is trained, the

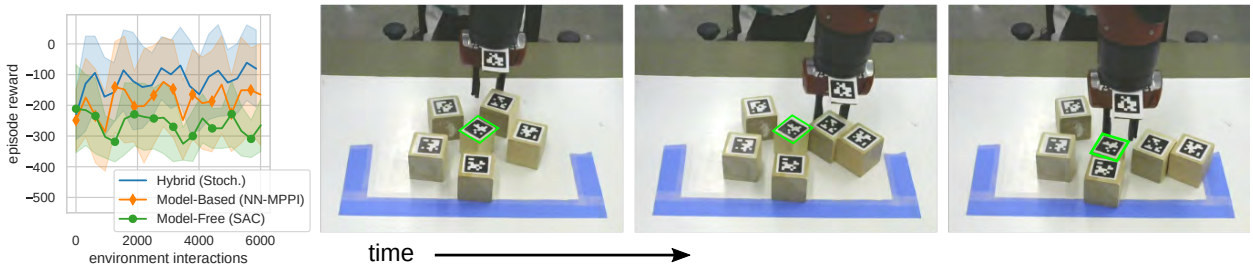

Figure 3–6 (Left) Hybrid learning Sawyer clutter results. Our stochastic hybrid learning method is able to achieve the task by effectively using both predictive models and model-free methods. Note: Markers are intended to distinguish between methods and do not represent all data points. Results are averaged over 10 random trials. **(Right) Time series task visualization.** The task is to access a target block (shown in green) surrounded by clutter (five other blocks) through environment interactions.

idea with this algorithm is that the model could be reused to learn another task with the same agent without needing to relearn a model, similar to the method we present here.

Fig. 3–5, shows our stochastic hybrid learning algorithm in comparison to other methods which combine model-based and model-free learning. Implementation details of the comparison methods can be found in Appendix A.2. These results show that our hybrid learning method outperforms these related methods. In addition to the learning gains, the hybrid learning controller maintains both the model and the policy after training. This may provide benefits during future controller execution—if the hybrid learning trained policy encounters an untested state during operation, it will have the model-based controller to fall back on to compensate for this uncertainty.

3.2.2 Hardware Experiment

We also apply hybrid learning to a Sawyer robot experiment to validate hybrid learning for real robot tasks. For this experiment, the goal is for the robot to access a block surrounded by clutter. The true positions of the "clutter" blocks and the "target" block are unknown to the robot. The robot's state is comprised of vectors from the robot's end-effector to each

block. The "target" block is always placed in the same start location, but the "clutter" blocks are randomly placed around the "target" block. The robot is rewarded for pushing the "clutter" blocks out of the way and accessing the "target" block (see Fig. 3–6 for task illustration). What makes this task difficult is that the robot must learn to push the "clutter" blocks out of the way rather than directly reaching towards the "target" block. Since our method naturally relies on the predictive models when the policy is uncertain, the robot is able to plan through the clutter to achieve the task. Fig. 3–6 shows that our proposed hybrid learning approach out performs both the model-based and model-free methods. During testing, we observed that the model-based approach tries to directly "reach" for the block without first clearing the clutter, which often results in pushing the target block outside workspace. Although the model-free method rarely pushes the block outside the workspace, it takes significantly longer for the model-free method to discover the pushing dynamics. Our hybrid approach learns to push the clutter blocks out of the way before accessing the target block.

3.3 Discussion and Summary

This chapter presented hybrid learning as a method for formally combining model-based learning with model-free policy learning based on hybrid control theory. Our approach derives the best action a robotic agent can take given the learned models, both model-free and model-based. We tested our approach in various simulated and real-world environments using a variety of learning conditions and show that our method improves both the sample-efficiency and the resulting performance of learning motor skills.

Our robot arm experiment demonstrated the ability of agents to learn online in a relatively simple physical test environment. The state was simplified to include the locations of each object relative to the end-effector. Future work could represent the state as an image and extend hybrid learning to visuo-motor tasks. Or the state could be expanded to include additional sensors such as contact sensors, accelerometers, etc. Our hybrid learning framework enables extensive hardware testing–due to its real-time implementation–to determine how learning performance changes when adding new sensors into the state space and when modifying reward functions.

As computation and memory allow, future work could also apply hybrid learning to model-based control using ensembles. One potential approach could train an ensemble of models and use the model uncertainty to determine which model-based controller to fall back on when the policy is uncertain. In general, bringing the robustness benefits of techniques such as path integral control [**165**] to our method can only improve its overall performance, but there may be many approaches that would all be equally viable.

Finally, a limitation of this work is that exploration currently is done through random sampling. Future work could explore methods of incorporating exploration goals into the algorithm. As the state space grows, random sampling may no longer be sufficient to span the state space. Alternate forms of exploration could help reduce the number of candidate samples required and reduce computation time. In Chapter 4, we introduce a method to systematically incorporate exploration into RL policies.

Chapter 4

Decorrelating Reinforcement Learning Experiences

The assumption that data are independent and identically distributed (*i.i.d.*) underpins all statistical machine learning approaches. When data are collected sequentially from agent experiences this assumption does not generally hold. In this chapter, we derive a method that overcomes these limitations by exploiting the statistical mechanics of ergodic processes, which we refer to as Maximum Diffusion Reinforcement Learning (MaxDiff RL). By decorrelating agent experiences, our approach provably enables single-shot learning in continuous deployments over the course of individual task attempts. Moreover, we prove our approach generalizes well-known maximum entropy techniques and robustly exceeds state-of-the-art (SOTA) performance across popular simulation benchmarks.

4.1 Maximum Diffusion Theory

Most Reinforcement Learning (RL) methods—including techniques like Maximum Entropy Reinforcement Learning (MaxEnt RL) [**166**–**173**]—presume that taking random actions produces effective exploration [**174**,**175**]. However, whether or not this is true depends on the agent's controllability properties and the temporal correlations of their experiences.

Controllability is a formal property of control systems that describes their ability to reach arbitrary states in an environment [**176**, **177**]. When linearizable systems become uncontrollable, state transitions become degenerate and irreversibly correlated. However, if the agent is controllable, the impact of correlations can be overcome, at least in principle. In this chapter, we define controllability in the context of Markov Decision Processes (MDPs)[1] similar to [**168**, **170**, **178**].

Definition 4.1 (Controllability) *The state transition dynamics,* $p(x_{t+1}|x_t, u_t)$ *of an MDP* $(\mathcal{X}, \mathcal{U}, p, r, \gamma)$ *are fully controllable when there exists a policy* $\pi : \mathcal{U} \times \mathcal{X} \to [0, \infty)$ *such that:*

$$p_\pi(x_{t+1}|x_t) = \mathbb{E}_{u_t \sim \pi(\cdot|x_t)}[p(x_{t+1}|x_t, u_t)] \tag{4.1}$$

and

$$D_{KL}(p_\pi(x_{t+1}|x_t) \parallel \nu(x_{t+1}|x_t)) = 0, \quad \forall t \in \mathbb{Z}^+ \tag{4.2}$$

for any arbitrary choice of state transition probabilities, $\nu : \mathcal{X} \times \mathcal{X} \to [0, \infty)$*, where* $D_{KL}(\cdot \| \cdot)$ *denotes the Kullback-Leibler (KL) divergence.*

A system is *fully controllable* when it is simultaneously capable of reaching every state and controlling *how* each state is reached.

While the relationship between controllability and temporal correlations has been studied for decades [**179**], only recently have researchers begun to study its impact on learning processes [**180**–**182**]. The ability to avoid temporal correlations and their impact depends on the properties of the underlying agent-environment dynamics. Destroying all correlations between agent experiences requires the ability to discontinuously jump from state to state

[1] In this chapter, $\mathcal{S}, \mathcal{A}$ from the MDP formulation introduced in Section 2.1.2 are replaced with $\mathcal{X}, \mathcal{U}$ and the discount factor γ is added to the MDP formulation.

without continuity of experience. However, continuity of experience is essential to many RL problem domains because most physically embodied systems require smooth transition. Failure to mitigate temporal correlations between state transitions can prevent effective exploration, severely impacting the performance of deep RL agents.

We formulate the agent-environment state transition dynamics as an autonomous stochastic process, whose sample paths $x(t)$ take value in a state space $\mathcal{X} \subset \mathbb{R}^d$ at each point in time within an interval $\mathcal{T} = [t_0, t]$. Agent experiences are collections of random variables parameterized by time, whose realizations $x(t)$ are the sample paths of the underlying agent-environment process.[2] In this context, the probability density function over state trajectories, $P[x(t)]$, completely characterizes an agent's experiences.

We use maximum caliber variational optimization [**183**–**185**] as the framework for decorrelating agent experiences to enable effective exploration. Maximum caliber is a generalization of the variational principle of maximum entropy [**186**]. The goal of a maximum caliber variational optimization is to find the trajectory distribution $P_{max}[x(t)]$, which optimizes an entropy functional $S[P[x(t)]]$. The optimal distribution describes the agent paths with the least-correlated experiences. Without constraints, agents could sample states discontinuously and uniformly in a way that is equivalent to *i.i.d.* sampling, but this is not consistent with the continuous experiences of embodied agents.

To impose constraints on the optimization, we define the system's velocity fluctuations along sample paths $x(t)$ in the following way:

$$\langle \dot{x}(t)\dot{x}(t)^T \rangle_{x^*} = \int_{\mathcal{X}^\mathcal{T}} P[x(t)] \int_\mathcal{T} \dot{x}(\tau)\dot{x}(\tau)^T \delta(x(\tau) - x^*) d\tau \mathcal{D}x(t), \tag{4.3}$$

[2] When $\mathcal{T} = \{0, \cdots, T\}$ is discrete, we use $x_{0:T}$ instead of $x(t)$.

where $\delta(\cdot)$ denotes the Dirac delta function and $\langle\cdot\rangle$ is the statistical physics notation for an expectation (*i.e.*, $\mathbb{E}[\cdot]$). We assume the system's velocity fluctuations are not degenerate anywhere in the state space of the stochastic process, which guarantees our resulting path distribution is not degenerate.[3] The velocities of the trajectories in this expression should be interpreted as an integral representation of the stochastic differential equations describing the evolution of the sample paths of the system [**187**].

We define our measure of temporal correlations at every point in space $x^* \in \mathcal{X}$ over an interval of duration Δt in the following way:

$$\mathbf{C}[x^*] = \int_{t_i}^{t_i+\Delta t} K_{XX}(t_i, \tau) d\tau, \tag{4.4}$$

where $K_{XX}(t_1, t_2)$ is the autocovariance of $x(t)$ at pairs of samples in time evaluated over a chosen interval, $[t_i, t_i + \Delta t] \subset \mathcal{T}$, with a given $x(t_i) = x^*$. We can express our constraint as

$$\langle \dot{x}(t)\dot{x}(t)^T \rangle_{x^*} = \mathbf{C}[x^*], \quad \forall x^* \in \mathcal{X}. \tag{4.5}$$

These statistics are bounded everywhere in the exploration domain. We assume they satisfy Lipschitz continuity and their spatial variations are bounded.[4] To ensure a valid probability density over trajectories, we also require $P[x(t)]$ to integrate to 1.

We can express the complete variational optimization problem as

$$\begin{aligned} \underset{P[x(t)]}{\operatorname{argmax}} \; & - \underbrace{\int_{\mathcal{X}^\mathcal{T}} P[x(t)] \log P[x(t)] \mathcal{D}x(t)}_{\text{general maximum caliber objective}} - \lambda_0 \underbrace{\left(\int_{\mathcal{X}^\mathcal{T}} P[x(t)] \mathcal{D}x(t) - 1 \right)}_{\text{ensures valid probability densities}} \\ & - \underbrace{\int_{\mathcal{X}} Tr\Big(\Lambda(x^*)^T \Big(\langle \dot{x}(t)\dot{x}(t)^T \rangle_{x^*} - \mathbf{C}[x^*] \Big) \Big) dx^*}_{\text{constrains the local magnitude of the agent's velocity fluctuations}} , \end{aligned} \tag{4.6}$$

[3] If we constrained the system by directly bounding the magnitude of its velocities rather than the velocity fluctuations, we would not be able to guarantee the non-degeneracy of the resulting path distribution.

[4] Linearizability of the underlying agent dynamics is a sufficient condition to satisfy this property.

where λ_0 and $\Lambda(\cdot)$ are Lagrange multipliers. This constrained variational optimization admits an analytical solution for the maximum entropy path distribution. The derived optimal path distribution is

$$P_{max}[x(t)] = \frac{1}{Z} \exp\Big[-\frac{1}{2} \int_{t_0}^{t} \dot{x}(\tau)^T \mathbf{C}^{-1}[x(\tau)] \dot{x}(\tau) d\tau \Big], \tag{4.7}$$

where Z is a normalization constant. (4.7) describes the trajectories of an optimal agent with minimally correlated paths, subject to the constraints imposed by continuity of experience. Moreover, (4.7) is equivalent to the path distribution of an anisotropic, spatially-inhomogeneous diffusion process. Thus, minimizing correlations among agent trajectories leads to diffusion-like exploration, whose properties can be analyzed using statistical mechanics. This also means the sample paths of the optimal agent are Markovian and ergodic.

In Section 4.1.2, we use Birkhoff's ergodic theorem [**188**, **189**] to show ergodic sampling along continuous Markovian trajectories is the best possible alternative to *i.i.d.* sampling.

Theorem 4.1 (Birkhoff's Ergodic Theorem) *Let* $\{x_t\}_{t\in\mathbb{N}}$ *be an aperiodic and irreducible Markov process on a state space* $\mathcal{X}$ *with invariant measure* ρ *and let* $f : \mathcal{X} \to \mathbb{R}$ *be any measurable function with* $E[|f(x)|] < \infty$. *Then, one has*

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} f(x_t) = \mathbb{E}_{x_0\sim\rho}[f(x_0)] \tag{4.8}$$

almost surely.

In other words, Birkhoff's ergodic theorem states that the time-average of any function of an ergodic Markov chain is equal to its ensemble average. Unlike other RL frameworks which assume the Markov property, MaxDiff RL enforces it as a property intrinsic to the optimal agent's path distribution.

4.1.1 MaxDiff Generalizes MaxEnt

When an agent's trajectories satisfy (4.7), we describe the agent as maximally diffusive. However, agents do not realize maximally diffusive trajectories spontaneously. Doing so requires finding a policy capable of satisfying maximally diffusive path statistics, which forms the foundation of what we term MaxDiff RL. While any given policy induces a path distribution, finding policies that realize maximally diffusive trajectories requires optimization and learning. To this end, we define:

$$\begin{aligned} P_\pi[x_{0:T}, u_{0:T}] &= \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\pi(u_t|x_t) \\ P^r_{max}[x_{0:T}, u_{0:T}] &= \prod_{t=0}^{T-1} p_{max}(x_{t+1}|x_t)e^{r(x_t,u_t)} \end{aligned} \tag{4.9}$$

where we discretized the distribution in (4.7) as $p_{max}(x_{t+1}|x_t)$ and analytically rederived the optimal path distribution under the influence of a reward landscape, $r(x_t, u_t)$. Given the distributions in (4.9), the goal of MaxDiff RL can be framed as minimizing the KL divergence between them—that is, between the agent's current path distribution and the maximally diffusive one.

Since our maximum entropy functional is an expression over all possible trajectories, we define a potential that correctly expresses the "free energy" over possible system realizations. To this end, we define our potential over $\mathcal{T} = [t_0, t]$ as

$$\langle r[x(t)]\rangle_P = \int_{\mathcal{X}^\mathcal{T}} P[x(t)] \int_{t_0}^{t} r[x(\tau)] d\tau \mathcal{D}x(t), \tag{4.10}$$

which captures the average reward over all possible system paths. Formally, we must assume that $\langle r[x(t)]\rangle_P$ is bounded, which in practice is the case for policies and controllers derived

from these principles. Our free energy functional objective is

$$\underset{P[x(t)]}{\operatorname{argmin}} \langle r[x(t)]\rangle_P - S[P[x(t)]], \tag{4.11}$$

where we use $S[P[x(t)]]$ as a short-hand for the argument to (4.6). To find the optimal path distribution, we take the variation of (4.10) with respect to $P[x(t)]$ and integrate it into the optimal path distribution. The resulting minimum free energy path distribution is

$$P^r_{max}[x(t)] = \frac{1}{Z}\exp\Big[-\int_{t_0}^{t}\Big(r[x(\tau)] + \frac{1}{2}\dot{x}(\tau)^T\mathbf{C}^{-1}[x(\tau)]\dot{x}(\tau)\Big)d\tau\Big], \tag{4.12}$$

which represents the path distribution of a diffusion process in a potential field. The optimal directed exploration strategy scales the strength of diffusion with respect to the desirability of the state. In this sense, the potential biases the diffusion process. We refer to systems which satisfy these statistics as *maximally diffusive with respect to the underlying potential*.

To draw connections to the broader MaxEnt RL literature, we recast the KL divergence formulation of MaxDiff RL as an equivalent stochastic optimal control problem. In this framework, the goal is to find a policy that maximizes the expected cumulative rewards, so we can express the MaxDiff RL objective as

$$\pi^*_{\text{MaxDiff}} = \underset{\pi}{\operatorname{argmax}}\ \mathbb{E}_{(x_{0:T},u_{0:T})\sim P_\pi}\left[\sum_{t=0}^{T-1}\gamma^t\hat{r}(x_t,u_t)\right], \tag{4.13}$$

with $\gamma \in [0,1)$ and modified rewards given by

$$\hat{r}(x_t,u_t) = r(x_t,u_t) - \alpha\log\frac{p(x_{t+1}|x_t,u_t)\pi(u_t|x_t)}{p_{max}(x_{t+1}|x_t)}, \tag{4.14}$$

where $\alpha > 0$ is a temperature-like parameter we introduce to balance diffusive exploration and reward exploitation, as we discuss in the following section. With these results in hand, we may now state one of our main theorems.

Theorem 4.2 (MaxDiff RL Generalizes MaxEnt RL) *Let the state transition dynamics due to a policy* π *be* $p_\pi(x_{t+1}|x_t) = \mathbb{E}_{u_t\sim\pi}[p(x_{t+1}|x_t, u_t)]$. *If the state transition dynamics are assumed to be decorrelated, then the optimum of (4.13) is reached when* $D_{KL}(p_\pi \parallel p_{max}) = 0$ *and the MaxDiff RL objective reduces to the MaxEnt RL objective.*

Proof. Our goal in this proof is to take the MaxDiff RL objective function in (4.13) and explore its relationship to the MaxEnt RL objective. We start by modifying the objective in (4.13) to instead minimize a loss function $\hat{l}(x_t, u_t)$ with $l(x_t, u_t) = -r(x_t, u_t)$. Without loss of generality, we drop the discount factor γ and manipulate the objective function:

$$\begin{aligned}
&\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N}\hat{l}(x_t, u_t)\right] \\
&\quad=\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t, u_t)+\alpha\log\frac{p(x_{t+1}|x_t, u_t)\pi(u_t|x_t)}{p_{max}(x_{t+1}|x_t)}\right] \\
&\quad=\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t, u_t)\right]+\sum_{t=1}^{N}\mathbb{E}_{(x_t,u_t)\sim p,\pi}\left[\alpha\log\frac{p(x_{t+1}|x_t, u_t)\pi(u_t|x_t)}{p_{max}(x_{t+1}|x_t)}\right] \\
&\quad=\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t, u_t)+\alpha\log\pi(u_t|x_t)\right]+\sum_{t=1}^{N}\mathbb{E}_{(x_t,u_t)\sim p,\pi}\left[\alpha\log\frac{p(x_{t+1}|x_t, u_t)}{p_{max}(x_{t+1}|x_t)}\right]. \quad (4.15)
\end{aligned}$$

We have rearranged the terms in the MaxDiff RL objective by taking advantage of the linearity of expectations and the definition of P_π in (4.9). Next, we apply Jensen's inequality to the last term of our expression above—bringing the expectation over control actions into the logarithm—and do more algebraic manipulations:

$$
\begin{aligned}
&\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N}\hat{l}(x_t,u_t)\right] \\
&\quad\leq\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t,u_t)+\alpha\log\pi(u_t|x_t)\right]+\sum_{t=1}^{N}\mathbb{E}_{x_t\sim p}\left[\alpha\log\frac{\mathbb{E}_{u_t\sim\pi}[p(x_{t+1}|x_t,u_t)]}{p_{max}(x_{t+1}|x_t)}\right] \\
&\quad\leq\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t,u_t)+\alpha\log\pi(u_t|x_t)\right]+\sum_{t=1}^{N}\mathbb{E}_{x_t\sim p}\left[\alpha\log\frac{p_\pi(x_{t+1}|x_t)}{p_{max}(x_{t+1}|x_t)}\right] \\
&\quad\leq\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t,u_t)+\alpha\log\pi(u_t|x_t)+\alpha D_{KL}\Big(p_\pi(x_{t+1}|x_t)\Big|\Big|p_{max}(x_{t+1}|x_t)\Big)\right], \qquad (4.16)
\end{aligned}
$$

where we note that $\mathbb{E}_{u_t\sim\pi}[p_{max}(x_{t+1}|x_t)] = p_{max}(x_{t+1}|x_t)$ and use the definition of $p_\pi(x_{t+1}|x_t)$ from (4.1). Finally, we must show that the MaxEnt RL objective emerges from the MaxDiff RL objective under the assumption that an agent's state transitions are decorrelated. Completely decorrelating the state transitions of an agent in general requires the ability to arbitrarily jump between states (*i.e.*, full controllability, see Definition 4.1). Given full controllability, the optimum of (4.16) is also reached when $D_{KL}(p_\pi \parallel p_{max}) = 0$.

Applying the assumption of decorrelated state transitions not only simplifies (4.16) by removing the KL divergence term but also by saturating Jensen's inequality, which recovers the equality between the left and right hand sides of our equations:

$$
\mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N}\hat{l}_c(x_t,u_t)\right] = \mathbb{E}_{P_\pi}\left[\sum_{t=1}^{N} l(x_t,u_t)+\alpha\log\pi(u_t|x_t)\right], \qquad (4.17)
$$

where the subscript c indicates the assumption of decorrelated state transitions—either in the context of agents with continuous paths (with maximum diffusivity as a necessary condition) or in general (with full controllability as a sufficient condition).

We can now write down the simplified MaxDiff RL optimization objective with the added assumption of decorrelated state transitions:

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}_{(x_{1:N},u_{1:N})\sim P_\pi}\left[\sum_{t=1}^{N} \gamma^t \hat{l}_c(x_t, u_t)\right], \tag{4.18}$$

where we reintroduced γ and with

$$\hat{l}_c(x_t, u_t) = l(x_t, u_t) + \alpha \log \pi(u_t|x_t). \tag{4.19}$$

We can also write (4.18) as a maximization and replace the cost with a reward function:

$$\hat{r}_c(x_t, u_t) = r(x_t, u_t) + \alpha \mathcal{H}(\pi(u_t|x_t)), \tag{4.20}$$

where we replaced the energy notation $S[\pi(u_t|x_t)]$ with $\mathcal{H}(\pi(u_t|x_t))$, to highlight connections to other results in the literature. This objective is the MaxEnt RL objective [**190**], which proves that MaxDiff RL is a strict generalization of MaxEnt RL to agents with temporally correlated experiences. This also proves that maximizing policy entropy does not decorrelate state transitions in general because maximizing policy entropy does not minimize $D_{KL}(p_\pi \parallel p_{max})$. □

In contrast to MaxEnt RL, when the agent-environment state transition dynamics introduce temporal correlations, the MaxDiff RL objective continues to prioritize effective exploration by decorrelating state transitions and encouraging the system to realize maximally diffusive trajectories. As we have shown above, MaxEnt RL's strategy of decorrelating action sequences is only as effective as MaxDiff RL's strategy of decorrelating state sequences when the underlying system's dynamics do not introduce temporal correlations on their own.

4.1.2 Single-Shot Learning

As was introduced in Section 2.1.3, single-shot learning concerns learning in non-episodic environments over the course of a single task attempt. The ergodic properties of MaxDiff RL enable us to establish representation-agnostic guarantees on the feasibility of single-shot learning through the Probably Approximately Correct in Markov Decision Processes (PAC-MDP) learning framework [**86**, **88**]. We briefly introduced the PAC-MDP framework in Section 2.3.1, but we provide an extended definition of Definition 2.5 here.

Definition 4.2 (PAC-MDP) *An algorithm $\mathcal{A}$ is said to be PAC-MDP if, for any $\epsilon > 0$ and $\delta \in (0,1)$, a policy π can be produced with poly$(|\mathcal{X}|, |\mathcal{U}|, 1/\epsilon, 1/\delta, 1/(1-\gamma))$ sample complexity that is at least ϵ-optimal with probability at least $1-\delta$. In other words, if $\mathcal{A}$ satisfies*

$$\Pr\Big(\mathcal{V}_{\pi^*}(x_0) - \mathcal{V}_{\pi}(x_0) \leq \epsilon\Big) \geq 1 - \delta \tag{4.21}$$

with polynomial sample complexity for all $x_0 \in \mathcal{X}$, where

$$\mathcal{V}_{\pi}(x_t) = \mathbb{E}_{p,\pi}\left[\sum_{n=0}^{\infty} \gamma^n r(x_{n+t}, u_{n+t}) \middle| x_t = x\right] \tag{4.22}$$

is the value function and $\mathcal{V}_{\pi^*}(x_t)$ is the optimal value function, then $\mathcal{A}$ is PAC-MDP.

This framework states that an algorithm $\mathcal{A}$ is PAC-MDP when it is capable of learning a policy with polynomial sample complexity that can get arbitrarily close to the optimal policy with arbitrarily high probability. The framework is representation-agnostic in the sense that, regardless of whether $\mathcal{A}$ involves any kind of neural network representation, any algorithm that satisfies Definition 4.2 is guaranteed to be at least ϵ-optimal. Given this definition of the PAC-MDP framework, we are now able to state our next formal result.

Theorem 4.3 (MaxDiff RL Agents Can Learn in Single-Shot Deployments) *If there exists a PAC-MDP algorithm $\mathcal{A}$ with policy π^{max} for the MaxDiff RL objective (4.13), then the Markov chain induced by π^{max} is ergodic, and any individual initialization of $\mathcal{A}$ will asymptotically satisfy the same ϵ-optimality as an ensemble of initializations.*

Proof. Given $\mathcal{A}$ is capable of producing an ϵ-optimal policy π^{max} and that $p_{\pi^{max}}(x_{t+1}|x_t) = \int_{\mathcal{U}} p(x_{t+1}|x_t, u_t)\pi^{max}(u_t|x_t)du_t$, we take $D_{KL}(p_{\pi^{max}} \parallel p_{max}) \approx 0$ for some choice of ϵ. Let $\{x_t\}_{t\in\mathbb{N}}$ be a Markov chain with state transition properties determined by $p_{\pi^{max}}(x_{t+1}|x_t)$. Because $D_{KL}(p_{\pi^{max}} \parallel p_{max}) \approx 0$, the Markov chain described by $p_{\pi^{max}}(x_{t+1}|x_t)$ is ergodic with invariant measure ρ [**191**]. We can manipulate the definition of PAC-MDP from Def. 4.2 to get the expression:

$$\Pr\Big(\mathcal{V}_{\pi^*}(x_0) - \mathcal{V}_{\pi^{max}}(x_0) \leq \epsilon\Big) \geq 1 - \delta \tag{4.23}$$

$$\mathbb{E}_{x_0\sim\rho}\Big[\mathbf{1}[\mathcal{V}_{\pi^*}(x_0) - \mathcal{V}_{\pi^{max}}(x_0) \leq \epsilon]\Big] \geq 1 - \delta, \tag{4.24}$$

where $\mathbf{1}[\cdot]$ denotes an indicator function. This expression means that being PAC-MDP is equivalent to being at least ϵ-optimal on average at least $100 \times (1-\delta)\%$ of episodes. Let

$$f(x_t) = \mathbf{1}[\mathcal{V}_{\pi^*}(x_t) - \mathcal{V}_{\pi^{max}}(x_t) \leq \epsilon] \tag{4.25}$$

be a bounded observable. Applying Birkhoff's ergodic theorem to (4.25) yields

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \mathbf{1}[\mathcal{V}_{\pi^*}(x_t) - \mathcal{V}_{\pi^{max}}(x_t) \leq \epsilon] = \mathbb{E}_{x_0\sim\rho}[\mathbf{1}[\mathcal{V}_{\pi^*}(x_0) - \mathcal{V}_{\pi^{max}}(x_0) \leq \epsilon]], \tag{4.26}$$

which proves that any individual initial condition will satisfy the ensemble average. In turn, we have

$$\begin{aligned} &\Pr\Big(\mathcal{V}_{\pi^*}(x_0) - \mathcal{V}_{\pi^{max}}(x_0) \leq \epsilon\Big) \geq 1 - \delta \\ &\qquad \implies \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \mathbf{1}[\mathcal{V}_{\pi^*}(x_t) - \mathcal{V}_{\pi^{max}}(x_t) \leq \epsilon] \geq 1 - \delta \end{aligned} \tag{4.27}$$

almost surely, which proves that an algorithm that is PAC-MDP during multi-shot (episodic) learning is guaranteed to be PAC-MDP during single-shot (non-episodic) learning if the underlying Markov chain induced by the policy is ergodic. □

This proof clarifies why ergodic sampling along continuous Markovian trajectories is the best possible alternative to *i.i.d.* sampling (via Theorem 4.1). As a result of ergodicity, the sampling statistics of any individual realization of an ergodic process are indistinguishable from *i.i.d.* sampling asymptotically, almost surely.

4.1.3 Learning Robustness

Robustness to random seeds and environmental randomizations is a highly desirable feature of deep RL agents [**192**–**194**]. However, guaranteeing such robustness is challenging because it requires modeling the impact of neural representations on learning outcomes. Nonetheless, we can provide representation-agnostic guarantees through the PAC-MDP learning framework [**86**, **88**].

Theorem 4.4 (MaxDiff RL Agents are Robust to Random Seeds) *If there exists a PAC-MDP algorithm $\mathcal{A}$ with policy π^{max} for the MaxDiff RL objective (4.13), then the Markov chain induced by π^{max} is ergodic and $\mathcal{A}$ will be asymptotically ϵ-optimal regardless of initialization.*

Proof. The proof of this theorem extends the proof of Theorem 4.3. Once again, let

$$f(x_t) = \mathbf{1}[\mathcal{V}_{\pi^*}(x_t) - \mathcal{V}_{\pi^{max}}(x_t) \leq \epsilon] \tag{4.28}$$

be an observable. Now, let $\{x_t\}_{t\in\mathbb{N}}$ and $\{x'_t\}_{t\in\mathbb{N}}$ both be ergodic Markov chains with identical transition kernels given by $p_{\pi^{max}}$, but with different initial conditions $x_0, x'_0 \in \mathcal{X}$. Then, since Birkhoff's ergodic theorem guarantees that the time-averages of observables

from $\{x_t\}_{t\in\mathbb{N}}$ and $\{x'_t\}_{t\in\mathbb{N}}$ will converge to the same unique ensemble average over the invariant measure ρ (Theorem 4.1), the following is true:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T}|f(x_t)-f(x'_t)|=0 \tag{4.29}$$

for any $x_0, x'_0 \in \mathcal{X}$ almost surely. This proves that any PAC-MDP algorithm is guaranteed to be robust to random seeds and environmental initializations if the underlying Markov chain induced by the policy is ergodic. □

In this section, we have proved MaxDiff RL generalizes MaxEnt RL, MaxDiff RL agents are capable of single-shot learning, and MaxDiff RL agents are robust to initial conditions and random seeds. These properties are a result of the deep connections between maximally diffusive trajectories and ergodicity. The experimental results in the following sections demonstrate these properties.

4.2 Methods

For the experiments in this chapter, we use model-agnostic objective with an analytical expression for the local path entropy,

$$\underset{\pi}{\operatorname{argmax}}\ \mathbb{E}_{(x_{0:T},u_{0:T})\sim P_\pi}\left[\sum_{t=0}^{T-1}\underbrace{r(x_t,u_t)}_{\text{task exploitation}}+\alpha\ \underbrace{\frac{1}{2}\log\det\mathbf{C}[x_t]}_{\text{diffusive exploration}}\right], \tag{4.30}$$

whose optimum realizes the same optimum as (4.13) and where we omitted γ. There are many ways to express the MaxDiff RL objective, each of which may have implementation-specific advantages. In this sense, MaxDiff RL is not a specific algorithm implementation but rather a general problem statement and solution framework, similar to MaxEnt RL. In the following

experiments, we compare SOTA model-based and model-free algorithms—Neural-Network Model Predictive Path Integral Control (NN-MPPI) [**2**] and Soft Actor Critic (SAC) [**11**], respectively—to our MaxDiff RL framework. Our MaxDiff RL implementation is identical to NN-MPPI except for the diffusive exploration term shown above. However, this simple modification can have a drastic effect on agent outcomes.

Our experimental results are divided into two parts. The first part focuses on a toy example which we developed to illustrate the contributions of MaxDiff RL, and the second part tests three standard simulated RL benchmarks. All learning experiments were run across 10 seeds, so for each task there are 10 policies per method. All differences between MaxDiff RL and comparisons in Figs. 4–1, 4–3, and 4–5 are statistically significant with $P < 0.05$ (see Appendix B).

We test the RL benchmarks in both the multi-shot and single-shot settings introduced in Section 4.1.2. For multi-shot experiments, episodic return curves illustrate the policies' learning progress across each of the 10 random seeds. After a fixed number of training episodes, we evaluated the final policies across 100 episodes with randomized initial conditions. The non-episodic nature of single-shot learning means that there are no true single-shot episodes to plot. Instead, we plot the windowed returns for each policy throughout training to mimic the multi-shot episodic return curves. Since single-shot policies are never re-initialized, we did not perform episodic evaluations. Instead, we used a task-specific performance measure (*i.e.*, distance traveled) sampled across the 10 runs of each task to perform statistical comparisons.

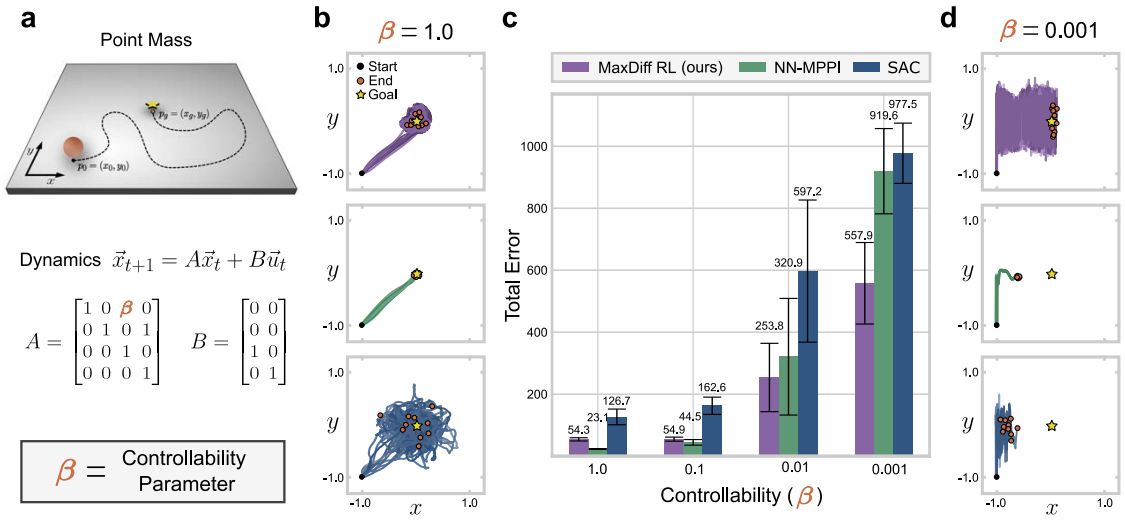

Figure 4–1 Point-mass Environment. **(a)** Illustration of the planar point-mass task and a definition of dynamics. We parameterize controllability through $\beta \in [0, 1]$, where $\beta = 0$ is uncontrollable. **(b, d)** Representative snapshots of MaxDiff RL, NN-MPPI, and SAC agents (top to bottom) in well-conditioned ($\beta = 1$) and poorly-conditioned ($\beta = 0.001$) controllability settings. **(c)**, Even in this simple system, poor controllability can break the performance of RL agents. As $\beta \to 0$ the system's ability to move in the x-direction diminishes, hindering the performance of NN-MPPI and SAC, while MaxDiff RL remains task-capable. Error bars represent the standard deviation from the mean across 100 evaluations of the final policy across 10 seeds each.

4.3 Point-mass Environment

To illustrate the impact temporal correlations can have on learning performance, we developed a planar point-mass task as a toy example. The task (shown in Fig. 4–1a) requires learning reward, dynamics, and policy representations from scratch to move a planar point-mass from a fixed initial position to a goal location within a fixed number of steps. The point-mass's true linear dynamics are simple enough to explicitly write down, and the policy admits a globally optimal analytical solution. The system's 4-dimensional state space contains planar positions and velocities. We parameterize controllability through $\beta \in [0, 1]$, which determines the relative difficulty of translating along the x-axis. When $\beta = 1$, it is equally

easy to move in x and y. As β decreases, it becomes more difficult to move in x. When $\beta = 0$, the point-mass is an uncontrollable system that can only translate along the y-axis.

4.3.1 Temporal Correlations Impact Performance

For most systems, controllability properties determine temporal correlations between state transitions. In Fig. 4–1, b and d illustrate representative snapshots of MaxDiff RL, NN-MPPI, and SAC agents (top to bottom) in well-conditioned ($\beta = 1$) and poorly-conditioned ($\beta = 0.001$) controllability settings. As expected, at $\beta = 1$ both NN-MPPI and SAC are able to accomplish the toy task (Fig. 4–1b). However, as $\beta \to 0$ the performance of NN-MPPI and SAC degrades parametrically until neither algorithm can solve the task (as shown in Fig. 4–1c). Hence, temporal correlations can hinder the learning performance of the SOTA in deep RL even in toy problem settings such as this one, where a globally optimal policy can be analytically computed in closed form. By contrast, our approach—MaxDiff RL—is able to consistently succeed at the task and is guaranteed to realize effective exploration by focusing instead on decorrelating agent experiences, (*i.e.*, state sequences, see purple curves in Fig. 4–1).

If we revisit the MaxEnt RL and MaxDiff RL objectives, we can see how the exploration terms relate to the point-mass dynamics. The MaxEnt RL objective (NN-MPPI & SAC) is

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{u_t \sim \pi}\left[\sum_{t=0}^{T-1} \underbrace{r(x_t, u_t)}_{\text{task exploitation}} + \alpha \underbrace{\mathcal{H}(\pi(u_t|x_t))}_{\text{policy entropy exploration}}\right], \tag{4.20}$$

and the MaxDiff RL objective is

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(x_t, u_t) \sim \pi}\left[\sum_{t=0}^{T-1} \underbrace{r(x_t, u_t)}_{\text{task exploitation}} + \alpha \underbrace{\frac{1}{2}\log\det \mathbf{C}[x_t]}_{\text{diffusive exploration}}\right]. \tag{4.30}$$

The point-mass dynamics are

$$x_{t+1} = Ax_t + Bu_t \quad \text{where} \quad A = \begin{bmatrix} 1 & 0 & \beta & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{4.31}$$

In the above equations, the colors emphasize which part of the dynamics each exploration term is modulating. MaxEnt RL explores the controls (shown in blue), indirectly exploring the states. MaxDiff RL explores the states (shown in purple) directly. Since β is part of the A matrix, it makes sense that exploring states results in more informative exploration that exploring controls. The real world systems we are interested in learning to control often have unknown controllability, so these results suggest that state exploration is better suited to real world learning.

4.3.2 Generalization Across Initialization Locations

During training, all point-mass episodes were initialized in the lower left corner of the environment as shown in Fig. 4–1. To assess the robustness of the learned policies to initialization locations, we tested the learned policies in four corners of the environment (three of which were unseen during training). By changing the initialization, we can assess how well policies learned the task—versus over-fitting to the training initialization. Each plot in Fig. 4–2 shows evaluations for 10 paths. β decreases from top to bottom. For NN-MPPI and SAC, we selected seeds that successfully learned the task during training. The columns display evaluations for different control methods. The results in Fig. 4–2 show that overall, MaxDiff RL state exploration during training results in better generalization to unseen initialization locations. Red boxes in Fig. 4–2 indicate examples of overfitting.

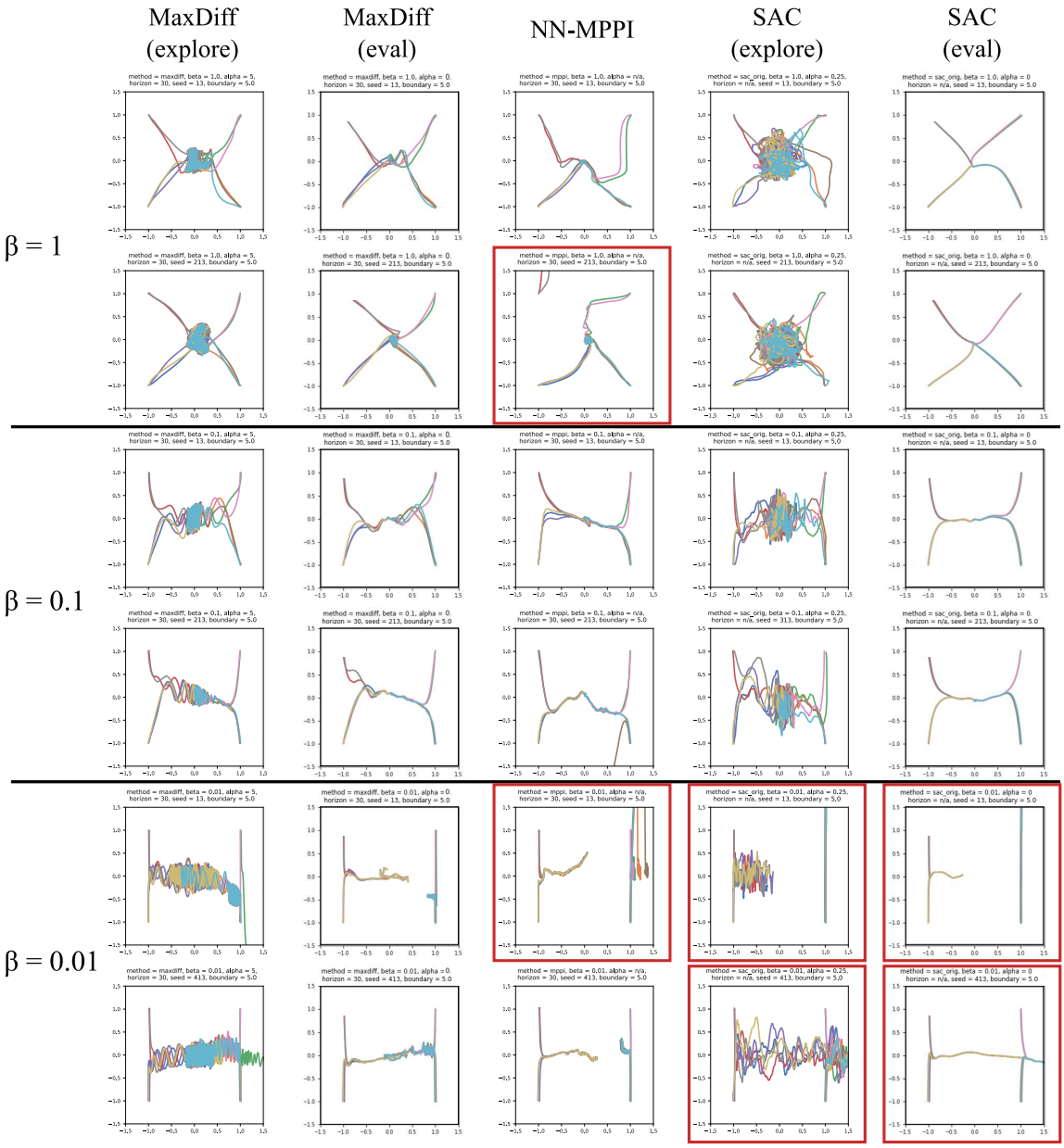

Figure 4–2 Sample Point-mass Evaluation Paths. All point-mass policies were trained with episodes initialized in the bottom left corner. Each plot shows 10 evaluations initialized from 4 corners. Rows show different β's as labeled on the left. Columns show different control methods. MaxDiff RL *explore* maintains the same α used during training, while *eval* sets $\alpha = 0$. SAC *explore* uses a control sampled from the policy, while *eval* uses the maximum likelihood (mean) control. All methods were trained for the same number of episodes. Red boxes indicate overfitting.

We also explored the potential benefits of including exploration during evaluation trials. We evaluated MaxDiff RL and SAC with and without their exploration terms. The first two columns show MaxDiff RL with the same α used during training (*i.e.*, with exploration) and $\alpha = 0$ (*i.e.*, without exploration) respectively. The last two columns show SAC with and without exploration. As β decreases, the footprint of the MaxDiff RL diffusion increases, indicating that the policy likely did not converge in the allotted training episodes. Therefore, exploration during evaluation helps MaxDiff RL reach states more quickly and ultimately solve the task for most initializations. The results also illustrate the differences between model-based and model-free paths. Even without exploration, there is some variability in the model-based (MaxDiff RL and NN-MPPI) executions due to the different model predictions, but SAC always executes exactly the same policy.

4.4 Simulated Benchmarks

Our benchmark experiments use the MuJoCo swimmer, ant, and half-cheetah environments. The goal of these environments is to achieve maximal $+x$ velocity. Additional details can be found in Appendix B.

4.4.1 Influence of the Temperature Parameter

The introduction entropy term in (4.30) requires MaxDiff RL agents to balance between two aims: achieving the task and embodying diffusion. While asymptotically there is no trade-off between maximally diffusive exploration and task exploitation, managing the relative balance between these two aims is important over finite time horizons, which we achieve

with a temperature-like parameter, α. For simple MaxDiff RL agents with fixed controllability properties in a reward landscape, α effectively sets the temperature of a heat bath induced by the policy. If α is too large, the system's fluctuations can overpower the influence of the reward and break ergodicity, which has been shown in the context of diffusion processes in potential fields [**195**]. Predicting the value of α and where this critical threshold lies is challenging due to its conceptual ties to ergodicity-breaking in nonequilibrium processes [**196**]. Future work should consider developing automated procedures for annealing α. One such method will be introduced in Chapter 5.

Since ergodicity provides many of MaxDiff RL's desirable properties and guarantees, tuning the value of α is essential. In Fig. 4–3, we explore the effect of tuning α on the learning performance. In the left plot, we vary α across multiple orders of magnitude and examine its impact on the terminal returns of MaxDiff RL swimmer agents. As we increase α from 1 to 100, we observe that diffusive exploration leads to greater returns. However, after $\alpha = 100$ we cross the critical threshold beyond which the system's diffusive exploration term overpowers the reward (see inset dotted line in Fig 4–3), thereby breaking the ergodicity of our agents with respect to the underlying potential and performing poorly at the task—as predicted by our theoretical framework.

4.4.2 Robustness to Initialization

Given a constant temperature of $\alpha = 100$ that preserves the swimmer's ergodicity, we compared the performance of MaxDiff RL to NN-MPPI and SAC across 10 seeds each. To ensure that the task was solvable by all agents, we tested swimmer bodies with two different tail masses; we refer to bodies with a tail mass $m_t = 1$ as heavy-tailed, and bodies with a

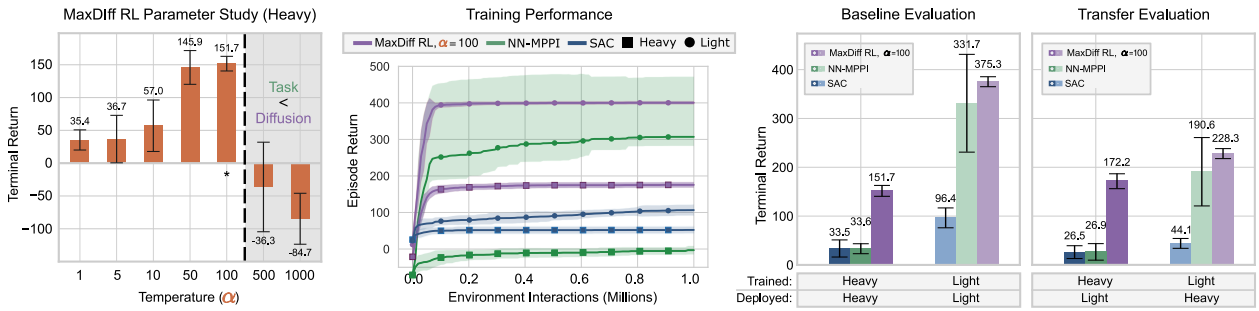

Figure 4–3 Swimmer benchmark results (10 seeds). **(left)** The terminal returns across values of α. Diffusive exploration leads to greater returns until a critical point (inset dotted line), after which the agent starts to maximize diffusing over accomplishing the task. **(middle)** Episode returns over time by method. Solid lines are the mean and shaded regions are the standard deviation. Markers distinguish two swimmer variants: one with a heavy-tail (all links have equal mass) and one with a light-tail (tail is 10% mass of other links). MaxDiff RL outperforms comparisons on average with near-zero variability across random seeds. **(right)** Average evaluated reward by method for 100 trials per seed. Error bars show standard deviation from the mean. As a baseline, we evaluate the learned representations on the trained swimmer variant. Then, we carry out a transfer experiment where the trained and deployed swimmer bodies are swapped for evaluation.

tail mass $m_t = 0.1$ as light-tailed.[5] The results of these methods are shown in the middle plot of Fig. 4–3. For the heavy-tailed swimmer, as MaxDiff RL is the only approach that takes into account temporal correlations, it is the only method to achieve the task. For the light-tailed swimmer, SAC struggles to succeed at this task within a million environment interactions, and NN-MPPI achieves good performance but with high variance across seeds. By contrast, MaxDiff RL's performance is nearly identical and competitive across all random seeds. By decorrelating state transitions, our agent was able to exhibit robustness to seeds and environment randomization beyond what is typically possible in deep RL. Moreover, since our MaxDiff RL implementation augments NN-MPPI, we can attribute performance gains and robustness to the properties of MaxDiff RL's theoretical framework.

[5] Other two link masses are $m = 1$.

4.4.3 Generalization Across Body Morphologies

To explore this possibility that agents are capable of finding optimal policies because their dynamics become indistinguishable from an ergodic diffusion process, we devised a transfer experiment. For the transfer experiments, we deployed the learned policies from the heavy-tailed and light-tailed agents in environments with swimmers of the opposite body morphology (which was not seen during training). The results of this experiment are shown on the right side of Fig. 4–3, where the results are categorized as "baseline" if the trained and deployed swimmer variants match, or "transfer" if they were swapped. First, we note that for both NN-MPPI and SAC representation transfer leads to degrading performance across the board. This is the case even when the swimmer variant they were deployed onto was more controllable. By contrast, MaxDiff RL agents can benefit from training on less controllable variants and improve their performance when deployed on the more controllable swimmer variant, as desired (see "Heavy-to-Light" transfer in Fig. 4–3). In other words, as the task becomes easier, we can expect the performance of MaxDiff RL agents to improve.

A more surprising result is the performance increase in MaxDiff RL agents between the baseline heavy-tailed swimmer and the "Light-to-Heavy" transfer swimmer in Fig. 4–3. We found that training with a more controllable swimmer increased the performance of agents when deployed on a heavy-tailed swimmer, showing that system controllability during training matters more to overall performance than the particular body morphology of the deployed system. This kind of *zero-shot* generalization [**39**] from an easier task to a more challenging task is reminiscent of results seen in RL agents trained via curriculum learning [**38**], as well as of the incremental learning dynamics of biological systems during motor skill acquisition more broadly [**197**]. However, here it emerges spontaneously from the

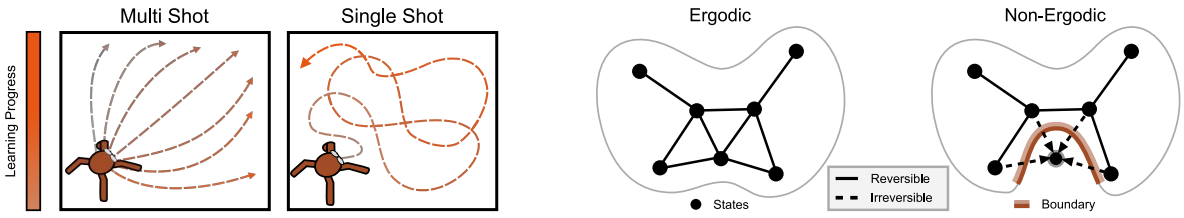

Figure 4–4 Single-shot Ant with irreversible state transitions. (**left**) Typically, RL algorithms learn across many different initializations and deployments of an agent, which is known as multi-shot learning. By contrast, single-shot learning insists on a single task attempt, which requires learning through continuous deployments. (**right**) Single-shot learning depends on the ability to generate data samples ergodically, which MaxDiff RL guarantees when there are no irreversible state transitions in the environment.

properties of MaxDiff RL agents. In part, this occurs because greater controllability leads to improved exploration, which increases the diversity of data observed during training. While formalizing this result is challenging, we note that MaxDiff RL encourages generalizable policies by focusing on agent outcomes instead of input actions. By forcing agent paths to match the statistics of an ergodic diffusion process, MaxDiff RL implicitly minimizes the effect of agent dynamics on performance.

4.4.4 Extension to Single-Shot Learning

Despite the challenges associated with studying the behavior of agents based on neural network representations, the ergodic properties of MaxDiff RL enables one to provide representation-agnostic guarantees on the feasibility of single-shot learning through the PAC-MDP learning framework. Thus, any MaxDiff RL agent capable of solving a task in a multi-shot fashion (in the PAC-MDP sense) is equivalently capable of solving the same task in a single-shot fashion. Since any two MaxDiff RL agents will asymptotically achieve identical learning outcomes, any individual MaxDiff RL agent will also achieve identical learning outcomes as an ensemble. Because ergodicity is the underlying principle behind the theory, we

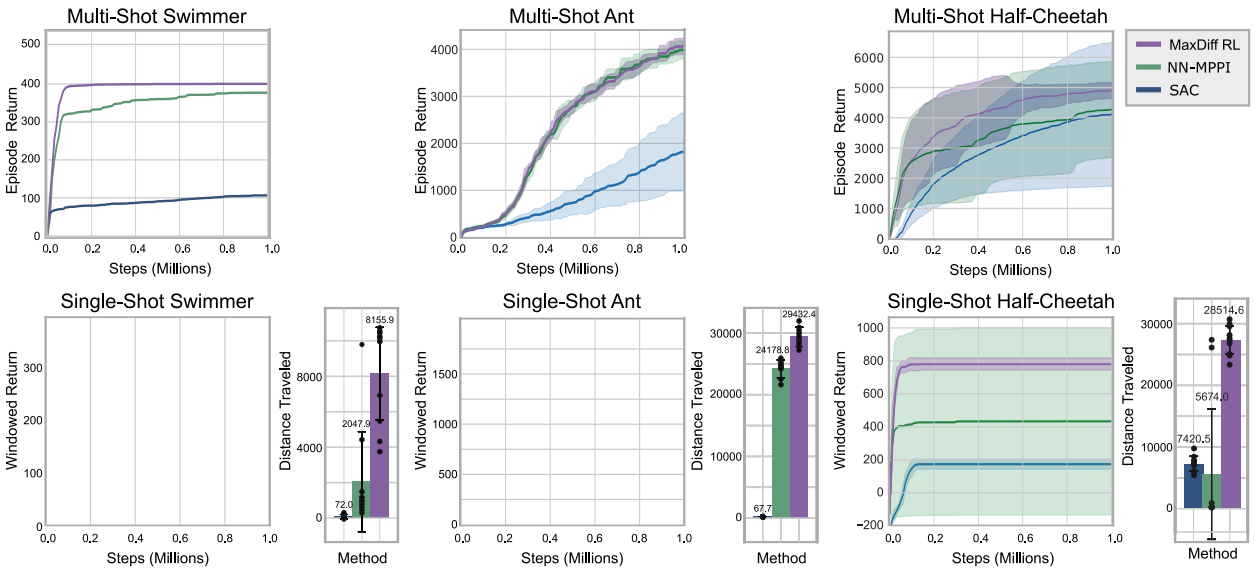

Figure 4–5 Simulated benchmark results across environments (10 seeds). The top row shows multi-shot results and the bottom row shows single-shot results. For all reward curves, solid lines are the average episode return (or windowed return over 1000 steps for single-shot) and shaded regions are the standard deviation. For all bar charts, values above each error bar are the mean and each error bar represents the standard deviation from the mean with n = 1000 (100 evaluations over 10 seeds for each condition). Here, we show that MaxDiff RL agents are equivalently capable of single-shot and multi-shot learning in a broad variety of settings. For the light-tailed swimmer, MaxDiff RL achieves robust performance comparable to its multi-shot counterpart. The ant and half-cheetah environments contain irreversible state transitions (*e.g.*, flipping upside down), which prevent ergodic trajectories. Nonetheless, MaxDiff RL remains SOTA across these environments.

expect its guarantees to fail when ergodicity is broken by either the agent or the environment. Fig. 4–4 provides a side-by-side comparison of multi-shot and single-shot deployments and illustrates ergodicity-breaking irreversible state transitions.

Figure 4–5 demonstrates the single-shot learning capabilities of MaxDiff RL agents, and explores what happens when ergodicity is broken by the properties of the environment. The primary difference between the swimmer and the other two environments is the existence of irreversible state transitions that can violate the ergodicity requirement of our single-shot learning guarantees. Unlike the swimmer, the ant and the half-cheetah are capable

of transitioning into such states by flipping upside down, thereby breaking ergodicity. Irreversible state transitions are common in real-world applications because they can arise as a result of unsafe behavior, such as a robot breaking or malfunctioning during learning. While such transitions can be prevented in principle through the use of safety-preserving methods [**198**–**200**], we omit their implementation to illustrate our point. As expected, the MaxDiff RL single-shot swimmer is capable of learning in continuous deployments (see Fig. 4–5), retaining the same robustness of its multi-shot counterpart, and achieving similar task performance. Despite ergodicity-breaking in the single-shot ant and half-cheetah environments, MaxDiff RL still leads to improved outcomes over NN-MPPI and SAC. However, the loss of ergodicity leads to an increase in the variance of single-shot MaxDiff RL agent performance, as well as equivalent performance to NN-MPPI ant in multi-shot, which we expect as a result of our robustness guarantees no longer holding.

4.5 Discussion and Summary

In this chapter, we highlighted how RL is sensitive to the temporal correlations intrinsic to many sequential decision-making processes. We introduced MaxDiff RL—a framework based on the statistical mechanics of ergodic processes—to overcome these limitations. Our framework offers a generalization of MaxEnt RL and addresses many foundational issues holding back the field. The ergodicity of MaxDiff RL agents enables data acquisition that is indistinguishable from *i.i.d.* sampling, performance that is robust to seeds, and single-shot learning. Our work forms a starting point for scientific study of embodied RL—where falsifiable predictions can be made about agent properties and their performance.

A limitation of this work is that, candidate actions for model-based control are chosen through random sampling. As the state space grows, random sampling may no longer be sufficient to span the state space. Future work could explore methods of incorporating the MaxDiff RL objective into other RL algorithms. Additionally, control techniques capable of enforcing ergodicity in the face of environmental irreversibility are needed to guarantee desirable agent properties like robustness to random seeds in complex problem settings [**201**]. Finally, the results in this chapter required manual tuning α—temperature-like parameter we introduced to balance diffusive exploration and reward exploitation. Approaches grounded in statistical physics for tuning or annealing temperature-like parameters during learning will be necessary to achieve effective exploration without sacrificing agent performance [**202**]. In Chapter 5, we present an automated procedure for annealing α to enable hardware experiments with MaxDiff RL.

Chapter 5

NoodleBot: A Hardware Reinforcement Learning Benchmark

In this chapter, we present NoodleBot—a low-cost, untethered 3-link swimmer robot—as a hardware benchmark for Reinforcement Learning (RL) algorithms. Although RL has demonstrated impressive capabilities in simulation, real-world applications often reveal a performance gap. A key challenge of real-world deployment is ensuring robustness to complex or unmodeled physical phenomena, such as anisotropic friction during locomotion. The discrepancy between simulated and real-world performance underscores the need for hardware benchmarks to rigorously evaluate and improve RL algorithms for physical deployment.

In contrast to prior swimmer platforms, the design of NoodleBot is not based on principles like bioinspiration [**203**, **204**] or modularity [**205**, **206**]. Instead, we present a simple, open-source[1] design that captures many of the complexities inherent to physical systems for approximately $100.[2] NoodleBot is equipped with firmware to handle low-level control, receive commands from an external laptop, and communicate state information back to the laptop via Robot Operating System (ROS). In the following sections, we describe the robot and the learning environment, demonstrate three algorithms learning on the platform, and compare the hardware results to learning with a simulated swimmer.

[1] See `https://github.com/MurpheyLab/NoodleBot` for CAD models, firmware, and assembly instructions.
[2] Excluding the cost of the positioning system.

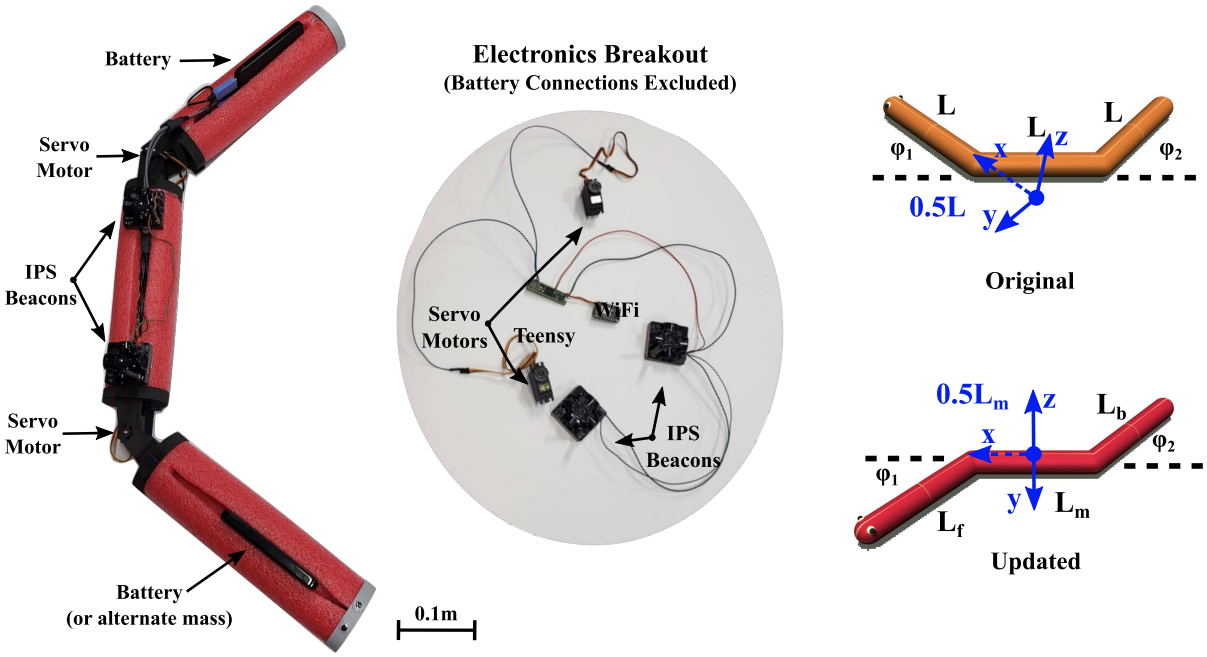

Figure 5–1 NoodleBot (Left) A top-down view of hardware. The total cost of the 3-link swimmer is approximately $100 excluding the cost of the Indoor Positioning System (IPS). (Middle) The electronic components housed inside the middle joint. (Right) Differences in joint orientations and body frames between the original simulated swimmer (top) and our hardware swimmer (bottom). The original body frame is offset $0.5L$ from the front link, where L is the link length. Our body frame is centered on the middle link. All joints shown in original and updated diagrams are 45°.

5.1 Robot Design

The robot swimmer contains three lightweight links with motorized hinge joints as shown in Fig. 5–1 and described in Table 5–1. The swimmer links are constructed using a pool noodle with slots cut out to accommodate batteries and electronics. The links are connected by custom hinged motor brackets 3D printed via Fused Deposition Modeling (FDM), and the ends are constrained by 3D printed FDM end caps. The bottom edge of each 3D printed part is flat (as opposed to round) to ensure good contact with the floor, and we attach felt pads to the flat portions to control friction. Standard 180° hobby servo motors (TowerPro SG-5010) provide actuation. Low-level robot control is handled on-board by a Teensy microcontroller

Link	Length	Mass	Components
Front	38cm	345g	Battery, FDM End Cap, FDM Bracket, Pool-Noodle Housing
Middle	43cm	495g	Servo Motors (2x), FDM Brackets (2x), WiFi Module, Teensy Microcotroller, Wiring, IPS Beacons (2x), Pool-Noodle Housing
Back	38cm	360g	Battery, FDM End Cap, FDM Bracket, Cables, Pool-Noodle Housing

Table 5–1 Details of NoodleBot Assembly by Link. Our robot is designed to be customizable. Length and mass values are provided as baselines but could easily be adjusted as desired (up to the torque capacity of the motors).

with a ESP8266 WiFi module attached, and standard commercial battery packs provide USB power (5V, 3A) to the motors, microcontroller, and WiFi module.

We use the Marvelmind IPS for robot state information. The IPS calculates the robot position using four stationary beacons that are placed around the test environment with line-of-sight to the robot. Then, two mobile beacons on the robot provide 2D state and rotation information via radio communication with a centralized modem.[3] The test area is roughly $3.4\text{m} \times 1.8\text{m}$ in x and y, respectively. With this configuration, the modem reports an update rate of 13.7Hz. Therefore, the robot takes several control steps per action command (similar to frame skips in MuJoCo). Alternate methods of collecting state information are possible (*e.g.*, April Tags), but the robot firmware may need to be modified to account for any differences in update rate or message type.

5.2 Learning Environment

NoodleBot is designed to mimic the Gym MuJoCo 3-link Swimmer [**13**]. The goal of our hardware swimmer is to learn a gait to maximize velocity in the $+x$ direction by applying torques to the joints in a constrained, planar environment. The hardware swimmer is able

[3] For these tests, we use firmware v7.900 for Super-Beacon-2 and Modem HW v5.1-2.

to run into the physical environment's boundaries in x and y. Each episode consists of 1000 control commands. We use the same reward function as the simulated environment,

$$r(s_t, a_t) = \frac{x_t - x_{t-1}}{dt} - 0.0001\|a\|^2, \tag{5.1}$$

where a are actions provided to the robot, s are robot states, x is the position of the robot in the world frame, $\|\cdot\|$ is the L_2 norm, t is a time index, and dt is the time step. The robot state contains the robot's world position (x, y, θ), joint angles (ϕ_1, ϕ_2), velocity in the world frame $(\dot{x}, \dot{y}, \dot{\theta})$, and joint angular velocities $(\dot{\phi_1}, \dot{\phi_2})$. The robot's world position is based on the midpoint of the two IPS beacons, which results in a different body frame from the original simulated swimmer (see Fig. 5–1).

Simulated and hardware swimmers differ in two primary ways. First, the hardware swimmer has to learn to operate in an environment with noisy sensors, impassable boundaries, unknown friction coefficients, stick-slip phenomena, degrading actuators, communication dropouts, as well as other real-world conditions that complicate the task. Second, the simulated swimmer operates in a highly viscous medium, whereas the hardware swimmer operates in air. As a result, differences in actuation and drag profiles lead to swimming gaits and policies that differ substantially between hardware and simulation—so much so that zero-shot policy transfer is not a viable strategy. A side-by-side comparison of the parameters for the simulated and hardware swimmers are listed in Table 5–2.

Property	Simulation	Hardware
State Dimension	10	11 *
Action Dimension	2	2
Time Step	$0.01s$	$0.05s$
Joint Limits	$\pm 100°$ §	$\pm 80°$ †
Joint Velocity Limits	None	± 0.3deg/ms †
Robot Angle Range	$\pm\infty$	$\pm\pi$
Reset Joint Angles	$\mathcal{U}(-0.1, 0.1)$	$\mathcal{U}(-1.4, 1.4)$
Reset Velocities (all)	$\mathcal{U}(-0.1, 0.1)$	0
Link Properties	Equal length and mass	See Table 5–1
Body Frame (Fig. 5–1)	Front link $+0.5L$ §	Center of middle link
Workspace Constraints	None	Barriers in x, y
Contact Friction	None	Unknown
Medium density	4000	$\sim$1.2 (air)
Medium viscosity	0.1	$\sim$0.00002 (air)

Table 5–2 Swimmer Environment Parameters Comparison. These parameters highlight differences between the simulated and hardware swimmer.

* Robot world angle θ is replaced with $[\sin(\theta), \cos(\theta)]$ to prevent angle wrapping discontinuities.

§ Simulated swimmer body frames and joint limits were updated to match our hardware frames.

† Joint angles and joint velocities are constrained to protect the motors.

5.3 Reinforcement Learning Benchmarks

We benchmark the performance of three deep RL algorithms on the simulated and hardware swimming tasks.[4] Two of the algorithms belong to the Maximum Entropy Reinforcement Learning (MaxEnt RL) family of techniques, which seek to decorrelate an agent's action sequences by maximizing policy entropy through the following objective,

$$\underset{\pi}{\text{argmax}}\ \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\sum_{t=1}^{T} r(s_t, a_t) + \alpha\mathcal{H}_\pi[a_t|s_t]\right], \tag{5.2}$$

[4] We tested the same algorithms as Chapter 4, but we describe them again for completeness.

where π is the policy, s is the robot state, a is the robot action, $\mathcal{H}_\pi[a_t|s_t]$ denotes the entropy of the policy conditioned on the current state, r is the predicted reward, and α is a temperature parameter. We use Soft Actor Critic (SAC) [**11**] to train a model-free MaxEnt policy, and we use Neural-Network Model Predictive Path Integral Control (NN-MPPI) [**2**] as our model-based MaxEnt planner.

The final algorithm is Maximum Diffusion Reinforcement Learning (MaxDiff RL) [**14**], which was introduced in Chapter 4. MaxDiff RL achieves exploration through decorrelating agent experiences (*i.e.*, explored states). We use the model-based MaxDiff RL formulation, which replaces $\mathcal{H}_\pi[a_t|s_t]$ from (5.2) with a state-transition based entropy term

$$S_\pi[s_t] = \frac{1}{2}\log\det C_\pi[s_t] \tag{5.3}$$

where $C_\pi[s_t]$ is the trajectory autocovariance under policy π. In this chapter, we additionally introduce an adaptive temperature parameter $\hat{\alpha}$ that modifies the original fixed temperature parameter from [**14**] by scaling α relative to the reward-scales of the paths sampled during each planning iteration. This allows the planner to account for the uncertainty of the learned transition and reward models. We define the adaptive temperature parameter as

$$\hat{\alpha} = \alpha \sum_{n=1}^{N} \frac{\left\| \sum_{t=1}^{T} r\left(s_t^{(n)}, a_t^{(n)}\right) \right\|}{\left\| \sum_{t=1}^{T} S\left[s_t^{(n)}\right] \right\|}, \tag{5.4}$$

where N is the number of sample paths, (n) indexes across sample paths, T is the planning horizon, S is the trajectory entropy from (5.3), r is the predicted reward, and α is a fixed temperature value. See the Table 5–3 for a summary of the learning hyperparameters.

Hyperparameters	**SAC [11]**	**NN-MPPI [2]**	**MaxDiff RL [14]**
Target Smoothing Coefficient	0.01	N/A	N/A
Discount Factor	0.99	0.95	0.95
Simulated Reward Scale	500 (Sim.), 10 (HW)	1	1
Planning Horizon	N/A	20	20
Planning Samples	N/A	1000	1000
Planning Inverse Temperature	N/A	0.1	0.1
Planning Control Noise	N/A	$\mathcal{N}(0, 0.5)$	$\mathcal{N}(0, 0.5)$
Diffusion Dimensions	N/A	N/A	$[x, y, \dot{x}, \dot{y}]$
Diffusion Weights (diagonal)	N/A	N/A	$[1, 1, 0.05, 0.05]$
Diffusion Temperature	N/A	N/A	0.1 (Sim.), 0.5 (HW)
MLP Hidden Layers	200×200 (ReLU Activation)		
MLP Final Layer Weights	$\mathcal{U}(-0.03, 0.03)$		
Learning Rate	0.0003 (Adam Optimizer [**207**])		
Batch Size	128 (5 updates per control step)		

Table 5–3 Learning Hyperparameters. Differences between simulation (Sim.) and hardware (HW) are noted above. Any hyperparameters not listed are the same as those in the original papers. MLP stands for multi-layer perceptron.

5.4 Results

All reinforcement learning algorithms were executed on an Intel® i7 CPU @ 2.60GHz x 12 laptop with an Nvidia® GeForce GTX 1650 Max-Q GPU. The laptop has Ubuntu 20.04, ROS Melodic, and Python 3.8 (PyTorch 1.11). We tested all three algorithms in simulation and on our hardware swimmer. For the simulated results in this section, we modified the body frames and joint limits from the original simulation to match our hardware frames (see Fig. 5–1). Each algorithm is trained with 5 different seeds (random model initializations) for 25 episodes of 1000 steps each. At the end of each episode, the model-based policies are reset, and the robot is physically reset to random joint angles at a fixed initial position in the world frame. During training, each algorithm completes five model updates per control step. Simulated and hardware tests take approximately 15 minutes and 1 hour, respectively.

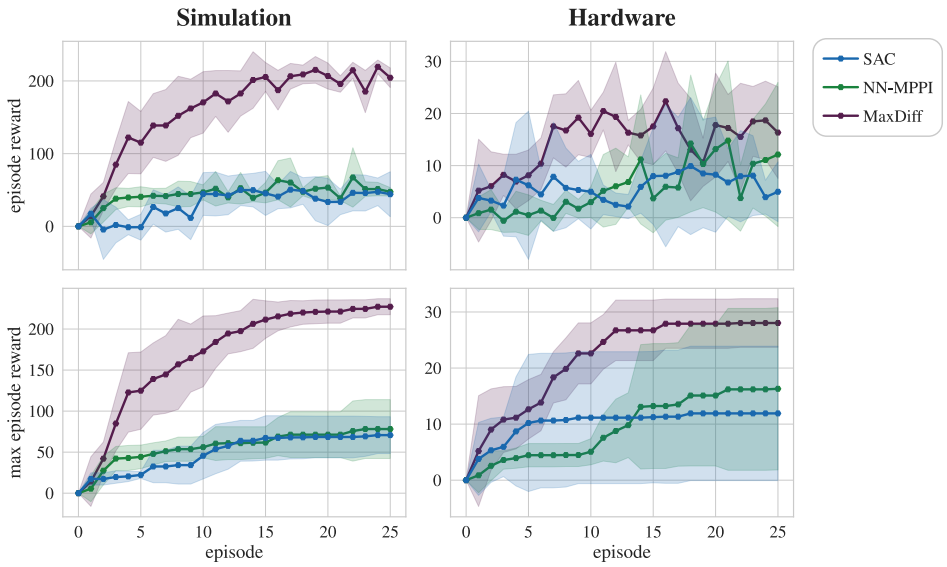

Figure 5–2 Learning Results. Each column shows results from the same tests (5 seeds, 25 episodes per method). Solid lines are the mean across seeds, and shaded regions are the standard deviation. **(Top Row)** Raw episode rewards across all episodes by method. MaxDiff RL outperforms SAC and NN-MPPI across in both simulation and hardware. **(Bottom row)** Cumulative maximum reward achieved by a given seed by method. MaxDiff RL seeds learn the task within 25 episodes, while there is large variance in the performance of SAC and NN-MPPI seeds.

First, we present results that illustrate the performance of the algorithms during training. Fig. 5–2 shows a side-by-side comparison of the episodic reward across training episodes of the simulated and hardware swimmers. The plots show that the algorithms perform similarly in both hardware and software and that MaxDiff RL outperforms NN-MPPI and SAC in both environments. To illustrate the ways in which the quality of RL policies relates to data collected, we also show agent sample paths for each seed's training episodes in the left section of Fig. 5–3. These paths indicate that when SAC and NN-MPPI collect diverse data, they are able to solve the task, but many seeds fail to explore very far beyond the initial position. MaxDiff RL, which is designed to directly encourage state exploration, indeed covers a much larger portion of the environment for all seeds in both software and hardware. As a result, MaxDiff RL attains better, more reliable performance across seeds and benchmarks, underscoring the importance of effective state exploration in RL.

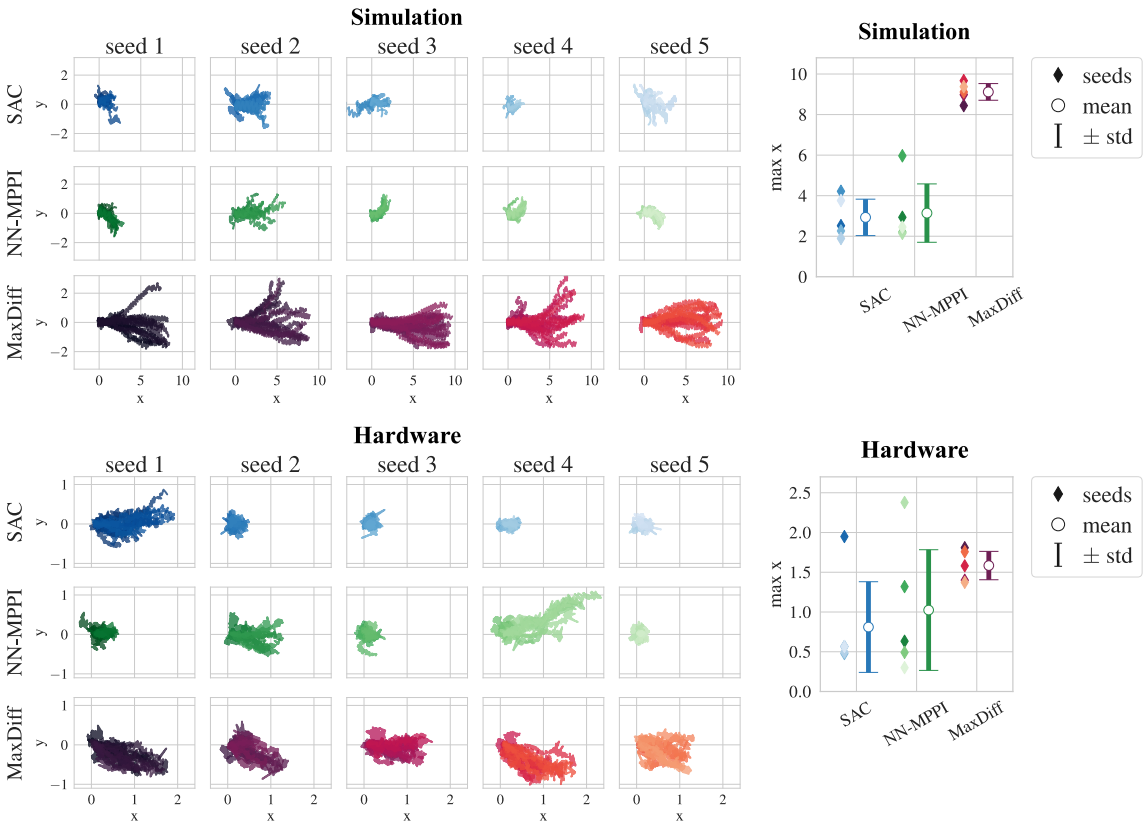

Figure 5–3 Training Paths by Seed. (Left) Exploration paths for simulation tests (top) and the hardware tests (bottom). All MaxDiff RL seeds spread out more than SAC and NN-MPPI paths in both simulation and hardware. (Right) Maximum x reached by each seed across all episodes. Diamonds are the raw data for each seed; error bars show the mean and standard deviation for each method. All MaxDiff RL seeds reach the same distance, regardless of the initialization, while there is large variance in the maximum distance reached by different SAC and NN-MPPI initialization.

Since the task is to maximize $+x$ velocity, the right section of Fig. 5–3 summarizes the maximum x position achieved by each seed. These results show that MaxDiff RL reaches similar final x positions across seeds. Thus, the empirical robustness of MaxDiff RL across random seeds in both software and hardware tests suggests that MaxDiff RL presents a promising approach to reliable RL in the real-world. Lastly, we note that additional tuning of the temperature parameter could lead to better overall performance.

After training, we evaluate each final policy 5 times. The results are summarized in Fig. 5–4. The simulated results show that all MaxDiff RL evaluations successfully solved

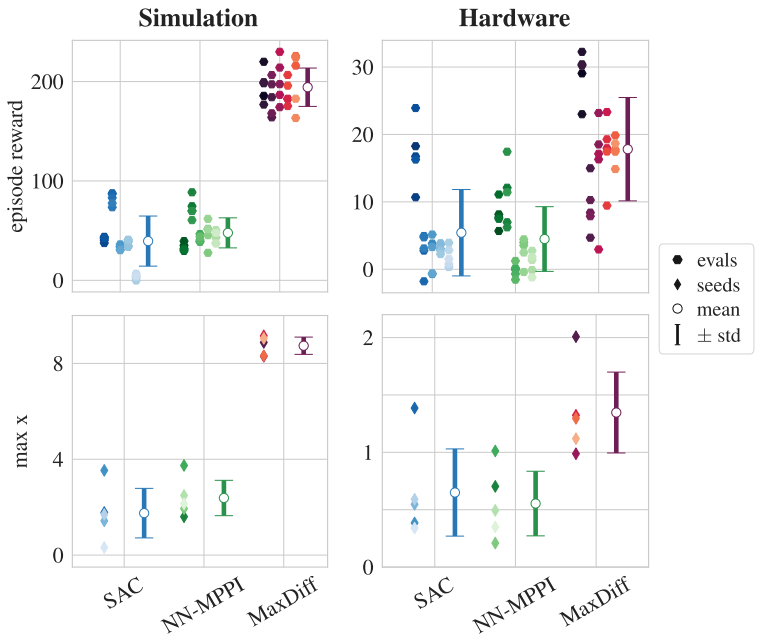

Figure 5–4 Eval Results. Each column shows results from the same tests (5 seeds, 5 evaluation episodes per method). Error bars show the mean and standard deviation. **(Top Row)** Episode reward for each evaluation episode. Hexagons show raw data by seed. **(Bottom Row)** Maximum x reached by each seed across episodes. Diamonds show raw data by seed. All simulated MaxDiff RL seeds receive a high reward and reach the same distance, regardless of the initialization, while there is large variance in the max distance reached by different SAC and NN-MPPI initializations. In hardware, MaxDiff RL outperforms SAC and NN-MPPI on average across episode reward and max x reached. As expected, the results are much noisier than the simulated results. In hardware, the robot has to contend with noisy sensors, physical boundaries, and wear on the system.

the task. SAC and NN-MPPI had lower episodic rewards, and both had larger variances in their simulated maximum x positions than MaxDiff RL. The hardware results show more variance than the simulated results, as expected. Nonetheless, MaxDiff RL still outperforms SAC and NN-MPPI on average, but additional temperature tuning may be required to achieve optimal performance. All policy evaluations were performed after all training, so it is likely that NoodleBot experienced actuator degradation between training and evaluation, which potentially explains the increase in variance. However, since hardware degradation is an unavoidable feature of real-world robot operation, these results show that our hardware swimmer is a good candidate for hardware reinforcement learning benchmarking.

5.5 Discussion

This chapter described NoodleBot as an open-source, low-cost 3-link swimmer robot platform designed to facilitate hardware experimentation in RL and to encourage the development of RL algorithms that directly grapple with the challenges of real-world operation. We demonstrated the performance of both model-based and model-free RL algorithms on this benchmark, providing baseline results for future comparisons and extensions.

The flexibility of our platform enables future work to explore various aspects of embodied operation, such as comparing the performance of learning algorithms across different body morphologies (*e.g.*, limb masses and lengths), testing in diverse environmental conditions (*e.g.*, different surfaces and interfacial physics), as well as assessing and adapting to robot wear and tear (*e.g.*, actuator degradation and communication dropouts). Moreover, due to the intrinsic stability of planar swimmer locomotion, NoodleBot presents a compelling platform for tasks requiring long planning horizons. Furthermore, NoodleBot is well-suited for evaluating other RL frameworks, including multi-objective RL and single-shot RL [**14**,**47**, **208**]. The platform's potential can be further expanded by incorporating low-cost sensors, such as cameras, to enable research in additional RL domains.

Beyond its research contributions, NoodleBot's affordability and open-source nature position it as a potentially valuable educational tool. Its accessible design and straightforward construction process can empower students and hobbyists to engage directly with the challenges and intricacies of real-world RL, similar to platforms like Duckietown [**209**]. By providing a tangible platform for implementing and testing RL algorithms, we hope NoodleBot can bridge the gap between theoretical understanding, simulation performance, and practical application, fostering a deeper appreciation for the complexities of embodied learning.

Chapter 6

Perception Learning for Unknown Objects with an Unknown Sensor

Deep generative learning has enabled large leaps in state-of-the-art (SOTA) performance on recognition tasks. These improvements have occurred alongside the widespread use of pre-existing datasets for training—most of which are based on images, videos, or text. For many interesting scenarios–such as performing tasks underwater or learning to use an ultrasound imager—there may be insufficient existing data to train an algorithm offline. Embodied agents, like robots, could fill this gap by simultaneously collecting data and training models online. Furthermore, it would be useful for a robot to be able to generate new representations on-the-fly either when it encounters new objects or more broadly if it determines that its current representation is insufficient. In such cases, agents can either collect training data through random sampling or plan to actively collect future data samples using an information measure. For training data to be suitable for many generative methods, the data must contain independent and identically distributed (*i.i.d.*) samples with respect to some underlying true distribution. This assumption generally does not hold for real-world problems in which agents build their training sets sequentially. Particularly because embodied agents

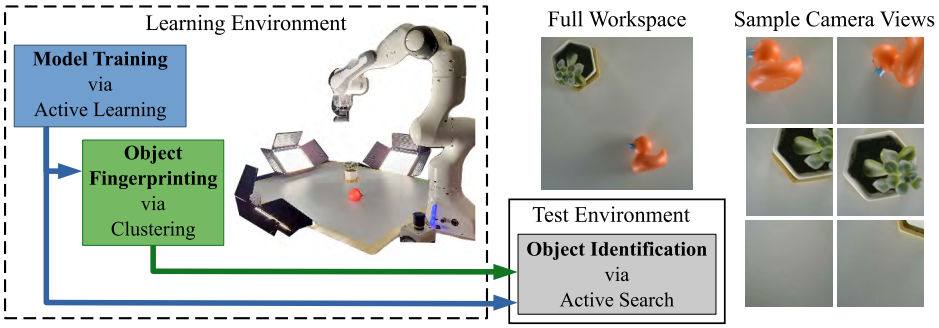

Figure 6–1 Learning Environment. All hardware experiments were performed with a webcam attached to a robot arm's end-effector. The goal is for the robot to explore a planar workspace while simultaneously building a representation of all observed objects in real-time. The learning process takes approximately 15 minutes to run 3000 exploration steps and 9000 model updates.

cannot move discontinuously, so trajectories are at least continuous and there is a real-world cost to movement.

In this chapter, we treat the robot as an *active learner* that has control over the data which is collected from the workspace. We apply ergodic stability and PAC-Bayes Theory to extend statistical guarantees for Variational Autoencoders (VAEs) to embodied agents. Then, we introduce our method for actively learning features of unknown objects and building representations of those objects for future tasks. We use object identification as an example task to evaluate our learned models and object representations. We apply our experimental platform to a planar environment with an RGB camera sensor and perform ablation experiments. Finally, we conclude with experimental demonstrations in higher dimensions and with additional sensors.

6.1 Embodied CVAE Learning Theory

As generative methods have grown in popularity, there have been an increasing number of papers that seek to understand the empirical success of these methods. Many works use the

Probably Approximately Correct (PAC) learning framework as a baseline for establishing theoretical guarantees under *i.i.d.* data assumptions [**80**–**84**]. While other related work focuses on establishing guarantees for non-*i.i.d.* data [**210**–**212**], we seek to collect data in a way that satisfies the data requirements of the learning algorithm. Our embodied problem necessitates a connection between the physical states of the robot and the sensor data—the model requires *i.i.d.* sensor data, and the robot requires states to travel to collect the sensor data. To connect these elements, we use a Conditional Variational Autoencoder (CVAE) as our learning representation, which was previously introduced in Section 2.2.2. The encoder conditional state connects sensor data and robot embodiment because each data point now has an associated encoded state. The decoder conditional state and decoded predicted variance connects possible data collection states to model uncertainty. We will discuss the role of the predicted variance further in Section 6.2.

To generate CVAE PAC-Bayes bounds, we follow a similar process to the VAE PAC-Bayes bounds outlined in Section 2.3.2. [**84**] provided an extension of PAC-Bayes to distributions with conditional priors with Assumption 2.1, which we restate here for section readability.

Assumption 2.1 (Conditional Posterior and Loss Function Properties) *A distribution* $y \mapsto q(\cdot|y)$ *and a loss function* ℓ *satisfy Assumption 2.1 with constant* $K > 0$ *if there exists a family* $\mathcal{E}$ *of functions* $\mathcal{H} \to \mathbb{R}$ *such that the following properties hold*

(1) The function $y \mapsto q(\cdot|y)$ *is continuous in the sense that for any* $y_1, y_2 \in \mathcal{Y}$,

$$d_{\mathcal{E}}(q(h|y_1), q(h|y_2)) \leq K d(y_1, y_2), \tag{6.1}$$

where $d_{\mathcal{E}}$ *are Integral Probability Metrics (IMPs)—see [**91**]—defined on the family of functions* $\mathcal{E}$ *and* d *is an underlying metric on* $\mathcal{Y}$.

(2) The function $\ell(\cdot, y) : \mathcal{H} \to \mathbb{R}$ *is in* $\mathcal{E}$ *for any* $y \in \mathcal{Y}$.

To incorporate the CVAE networks into Conditional PAC-Bayes Assumption 2.1, we need functional representations of the encoder and decoder CVAE networks. The primary change from the previous VAE definition is the addition of the conditional state as an input to each function. The CVAE functions are defined as follows:

Definition 6.1 (CVAE Encoder, Decoder, and Loss Functions) *Given Euclidean instance spaces $\mathcal{X}$ and $\mathcal{Y}$ and a latent space $\mathcal{Z} = \mathbb{R}^d$, let $q_\phi(z|x,y)$ be a Gaussian latent space distribution $\mathcal{N}(\mu_\phi(x,y), diag(\sigma^2_\phi(x,y)))$, where $\mu_\phi(x,y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ and $\sigma_\phi(x,y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d_{\geq 0}$. We define the encoder, decoder, and loss functions as*

$$Q_\phi(x,y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{2d}, \text{ where } Q_\phi(x,y) = \begin{bmatrix} \mu_\phi(x,y) \\ \sigma_\phi(x,y) \end{bmatrix}, \tag{6.2}$$

$$g_\theta(z,x) : \mathcal{X} \times \mathcal{Z} \to \mathcal{Y}, \text{ where latent space samples } z \sim q_\phi(z|x,y), \text{ and} \tag{6.3}$$

$$\ell^\theta(z,x,y) : \mathcal{Z} \times \mathcal{X} \times \mathcal{Y} \to [0,\infty) \text{ where } \ell^\theta(z,x,y) = \|y - g_\theta(z,x)\|, \tag{6.4}$$

where $\|\cdot\|$ denotes the L_2 norm.

Our first CVAE assumption updates Assumption 2.1 (1) to incorporate conditional states.

Assumption 6.1 (CVAE Encoder and Decoder Properties) *The encoder and decoder are Lipschitz-continuous w.r.t. their inputs meaning there exist real numbers $K_\phi, K_\theta > 0$ such that for any $x_1, x_2 \in \mathcal{X}$, $y_1, y_2 \in \mathcal{Y}$, and $z_1, z_2 \in \mathcal{Z}$,*

$$\|Q_\phi(x_1,y_1) - Q_\phi(x_2,y_2)\| \leq K_\phi \left(\|x_1 - x_2\| + \|y_1 - y_2\|\right), \tag{6.5}$$

and

$$\|g_\theta(z_1,x_1) - g_\theta(z_2,x_2)\| \leq K_\theta \left(\|z_1 - z_2\| + \|x_1 - x_2\|\right). \tag{6.6}$$

Assumption 2.1 is formulated generally for conditional distributions. We require a further statement to show that for the CVAE loss function $\ell^\theta(x, y, z)$ there is a family $\mathcal{E}$ for which the continuity assumption is satisfied with constant K. The proof relies on the Wasserstein distance [**213**], which we define prior to the lemma.

Definition 6.2 (Wasserstein distances) *Given empirical distributions P and Q with samples $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ respectively, the distance measure is*

$$W_p(P, Q) = \left(\frac{1}{n} \sum_{i=1}^{n} \|x_i - y_i\|^p \right)^{1/p}. \tag{6.7}$$

Given Normal distributions p, q with parameters $\mathcal{N}(\mu, diag(\sigma^2))$, the distance measure is

$$W_2(p, q)^2 = \|\mu_p - \mu_q\|^2 + \|\sigma_p - \sigma_q\|^2. \tag{6.8}$$

We can now prove that our CVAE functions satisfy Assumption 2.1. The major difference between the following theorem and Theorem 2.1 is that the inequality includes additional terms due the decoder being conditioned on x (additional terms boxed below).

Lemma 6.1 (CVAE Satisfies Assumption 2.1) *For a CVAE with parameters ϕ and θ, let $K_\phi, K_\theta \in \mathbb{R}$ be the Lipschitz norms of the encoder and decoder respectively. Then the variational distribution $q_\phi(z|x, y)$ satisfies Assumption 2.1 with*

$$d_{\mathcal{E}}(q_\phi(z|x_1, y_1), q_\phi(z|x_2, y_2)) \leq K_\phi K_\theta \left(\boxed{\|x_1 - x_2\|} + \|y_1 - y_2\| \right) + \boxed{K_\theta \|x_1 - x_2\|} \tag{6.9}$$

and

$$\ell^\theta(z, x, y) \in \mathcal{E} \text{ for any } x \in \mathcal{X} \text{ and } y \in \mathcal{Y}, \tag{6.10}$$

where $\mathcal{E} = \{f : \mathcal{Z} \times \mathcal{X} \to \mathbb{R} \text{ s.t. } \|f\|_{Lip} \leq K_\theta\}$.

Proof. In Definition 6.1, we defined $q_\phi(z|x,y)$ as a normal distribution and $Q_\phi(x,y) = \left[\mu_\phi(x,y), \sigma_\phi(x,y)\right]^T$. Since the Wasserstein-2 distance for normal distributions has a closed form solution and Q is Lipschitz continuous as defined in (6.5), we can define the following inequality

$$W_2(q_\phi(z|x_1,y_1), q_\phi(z|x_2,y_2)) \leq K_\phi \left(\|x_1 - x_2\| + \|y_1 - y_2\|\right). \tag{6.11}$$

Using the definition of the family of functions $\mathcal{E} = \{f : \mathcal{Z} \times \mathcal{X} \to \mathbb{R} \text{ s.t. } \|f\|_{\text{Lip}} \leq K_\theta\}$ and the Kantorovich duality, we obtain the IMP $d_\mathcal{E}$ as

$$d_\mathcal{E}(q_\phi(z|x_1,y_1), q_\phi(z|x_2,y_2)) = K_\theta(W_1(q_\phi(z|x_1,y_1), q_\phi(z|x_2,y_2)) + \|x_1 - x_2\|). \tag{6.12}$$

Since $W_1 \leq W_2$, combining (6.12) and (6.11) yields

$$d_\mathcal{E}(q_\phi(z|x_1,y_1), q_\phi(z|x_2,y_2)) \leq K_\phi K_\theta \left(\|x_1 - x_2\| + \|y_1 - y_2\|\right) + K_\theta \|x_1 - x_2\|. \tag{6.13}$$

Now we can examine the second part of Lemma 6.1. Let $\ell = \ell^\theta$, $x_1, x_2 \in \mathcal{X}$, $y \in \mathcal{Y}$, and $z_1, z_2 \in \mathcal{Z}$. We have

$$\begin{aligned}
\ell(z_1, x_1, y) - \ell(z_2, x_2, y) &= \|y - g_\theta(z_1, x_1)\| - \|y - g_\theta(z_2, x_2)\| \\
&= \|y - g_\theta(z_1, x_1) + g_\theta(z_2, x_2) - g_\theta(z_2, x_2)\| - \|y - g_\theta(z_2, x_2)\| \\
&\leq \|y - g_\theta(z_2, x_2)\| + \|g_\theta(z, x) - g_\theta(z_1, x_1)\| - \|y - g_\theta(z_2, x_2)\| \\
&= \|g_\theta(z_2, x_2) - g_\theta(z_1, x_1)\| \\
&\leq K_\theta \left(\|z_2 - z_1\| + \|x_2 - x_1\|\right)
\end{aligned} \tag{6.14}$$

where the first inequality uses the triangle inequality, and the second uses the Lipschitz assumption on g_θ. □

With the first part of the lemma, we have established that the decoder is continuous with respect to the IMP $d_{\mathcal{E}}$ and the underlying metrics $\mathcal{X}$ and $\mathcal{Y}$. We have also shown that the loss function captures the continuity of the decoder. We can use this lemma to prove a PAC-Bayes bound for the CVAE reconstruction loss. The proof relies on the Donsker-Varadhan change of measure proposition, which we state here.

Proposition 6.1 (Donsker-Varadhan Change of Measure) *Given probability measures p, q on a space $\mathcal{H}$ such that $q \ll p$, let $g : \mathcal{H} \to \mathbb{R}$ be a function such that $\mathbb{E}_{h\sim p} \exp(g(h)) < \infty$. Then,*

$$\mathbb{E}_{h\sim p} \exp\left(g(h)\right) \geq \exp\left(\mathbb{E}_{h\sim q}[g(h)] - D_{KL}(q\|p)\right), \tag{6.15}$$

where D_{KL} denotes the Kullback-Leibler (KL) divergence.

We can now prove a PAC-Bayes bound for the CVAE reconstruction loss. The major difference between the following theorem and Theorem 2.2 is that the inequality includes new terms due the decoder being conditioned on x (new terms boxed below). The second boxed term necessitates adding an additional term to the CVAE loss function.

Theorem 6.1 (CVAE PAC-Bayes Bounds) *Let $\mathcal{X}$ and $\mathcal{Y}$ be the instance spaces, $\pi \in \mathcal{M}_+^1(\mathcal{X})$ and $\zeta \in \mathcal{M}_+^1(\mathcal{Y})$ the data-generating distributions, Z the latent space, $p(z) \in \mathcal{M}_+^1(\mathcal{Z})$ the prior distribution, θ the decoder parameters, and $\delta \in (0,1)$, $\lambda > 0$ be real numbers. With probability at least $1-\delta$ over $S_x \overset{iid}{\sim} \pi$ and $S_y \overset{iid}{\sim} \zeta$, the following holds for any posterior $q_\phi(z|x,y)$:*

$$\begin{aligned}
\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim q_\phi(z|x,y)} \left[\ell^\theta(z,x,y)\right] \leq \frac{1}{n}\sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q_\phi(z|x_i,y_i)} \left[\ell^\theta(z,x_i,y_i)\right] + \frac{1}{\lambda}\Bigg[\sum_{i=1}^{n} D_{KL}(q_\phi(z|x_i,y_i)\|p(z)) \\
+ \log\frac{1}{\delta} + \frac{\lambda K_\phi K_\theta}{n}\sum_{i=1}^{n}\left(\boxed{\mathop{\mathbb{E}}_{x\sim\pi}\left[d(x,x_i)\right]} + \mathop{\mathbb{E}}_{y\sim\zeta}\left[d(y,y_i)\right]\right) + \boxed{\frac{\lambda K_\theta}{n}\sum_{i=1}^{n}\mathop{\mathbb{E}}_{x\sim\pi}\left[d(x,x_i)\right]} \\
+ n\log \mathop{\mathbb{E}}_{z\sim p(z)} \mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi} \mathop{\mathbb{E}}_{y'\sim\zeta}\left[\ell^\theta(z,x',y')\right] - \ell^\theta(z,x,y)\right)\right)\Bigg],
\end{aligned} \tag{6.16}$$

where K_ϕ, K_θ are encoder and decoder Lipschitz norms and $d(y,y') = \|y - y'\|$.

Proof. First, we consider a set $Z = \{z_1, \ldots, z_n\} \sim p(z)$ *i.i.d.* sampled from $p(z)$. By applying Markov's inequality to the positive random variable A, defined as

$$A \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \exp\left(\frac{\lambda}{n} \sum_{i=1}^{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z_i, x, y) \right] - \ell^{\theta}(z_i, x_i, y_i) \right)\right), \tag{6.17}$$

we obtain with probability at least $1-\delta$ over the draw of $S_x \overset{iid}{\sim} \pi, S_y \overset{iid}{\sim} \zeta, A \leq \frac{1}{\delta}\mathbb{E}[A]$, meaning

$$\begin{aligned} &\mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \exp\left(\frac{\lambda}{n} \sum_{i=1}^{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z_i, x, y) \right] - \ell^{\theta}(z_i, x_i, y_i) \right)\right) \leq \\ &\qquad \frac{1}{\delta} \mathop{\mathbb{E}}_{S_x \overset{iid}{\sim} \pi} \mathop{\mathbb{E}}_{S_y \overset{iid}{\sim} \zeta} \mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \exp\left(\frac{\lambda}{n} \sum_{i=1}^{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z_i, x, y) \right] - \ell^{\theta}(z_i, x_i, y_i) \right)\right). \end{aligned} \tag{6.18}$$

First, we focus on the left side of (6.18). Using algebraic manipulations and the Donsker-Varadhan change of measure theorem (Prop. 6.1) to get the final inequality, we get

$$\begin{aligned} &\mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \exp\left(\frac{\lambda}{n} \sum_{i=1}^{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z_i, x, y) \right] - \ell^{\theta}(z_i, x_i, y_i) \right)\right) \\ &\qquad = \mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \prod_{i=1}^{n} \exp\left(\frac{\lambda}{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z_i, x, y) \right] - \ell^{\theta}(z_i, x_i, y_i) \right)\right) \\ &\qquad = \prod_{i=1}^{n} \mathop{\mathbb{E}}_{z_i \sim p(z)} \exp\left(\frac{\lambda}{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z_i, x, y) \right] - \ell^{\theta}(z_i, x_i, y_i) \right)\right) \\ &\qquad = \prod_{i=1}^{n} \mathop{\mathbb{E}}_{z \sim p(z)} \exp\left(\frac{\lambda}{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z, x, y) \right] - \ell^{\theta}(z, x_i, y_i) \right)\right) \\ &\qquad \geq \prod_{i=1}^{n} \exp\Bigg(\mathop{\mathbb{E}}_{z \sim q(z|x_i, y_i)} \left[\frac{\lambda}{n} \left(\mathop{\mathbb{E}}_{x \sim \pi} \mathop{\mathbb{E}}_{y \sim \zeta} \left[\ell^{\theta}(z, x, y) \right] - \ell^{\theta}(z, x_i, y_i) \right)\right] \\ &\qquad\qquad\qquad -D_{KL}(q_{\phi}(z|x_i, y_i) \| p(z))\Bigg). \end{aligned}$$

By applying a logarithm and with additional algebraic manipulations, we get

$$\begin{aligned}
&\log \mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \exp\left(\frac{\lambda}{n}\sum_{i=1}^{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z_i,x,y)\right]-\ell^\theta(z_i,x_i,y_i)\right)\right)\\
&\quad\geq \log\prod_{i=1}^{n}\exp\Big(\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z,x,y)\right]-\ell^\theta(z,x_i,y_i)\right)\right]\\
&\qquad\qquad -D_{KL}(q_\phi(z|x_i,y_i)\|p(z))\Big)\\
&\quad=\sum_{i=1}^{n}\Big(\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z,x,y)\right]-\ell^\theta(z,x_i,y_i)\right)\right]\\
&\qquad\qquad -D_{KL}(q_\phi(z|x_i,y_i)\|p(z))\Big)\\
&\quad=\frac{\lambda}{n}\sum_{i=1}^{n}\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z,x,y)\right]-\ell^\theta(z,x_i,y_i)\right]\\
&\qquad\qquad -\sum_{i=1}^{n}D_{KL}(q_\phi(z|x_i,y_i)\|p(z)).
\end{aligned}\tag{6.19}$$

Similarly, we can use Fubini's theorem and algebraic manipulations on the right side of (6.18), such that

$$\begin{aligned}
&\mathop{\mathbb{E}}_{S_x\overset{iid}{\sim}\pi}\mathop{\mathbb{E}}_{S_y\overset{iid}{\sim}\zeta}\mathop{\mathbb{E}}_{Z\overset{iid}{\sim}p(z)}\exp\left(\frac{\lambda}{n}\sum_{i=1}^{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z_i,x,y)\right]-\ell^\theta(z_i,x_i,y_i)\right)\right)\\
&\quad=\mathop{\mathbb{E}}_{S_x\overset{iid}{\sim}\pi}\mathop{\mathbb{E}}_{S_y\overset{iid}{\sim}\zeta}\prod_{i=1}^{n}\mathop{\mathbb{E}}_{z\sim p(z)}\exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z,x,y)\right]-\ell^\theta(z,x_i,y_i)\right)\right)\\
&\quad=\prod_{i=1}^{n}\mathop{\mathbb{E}}_{x_i\sim\pi}\mathop{\mathbb{E}}_{y_i\sim\zeta}\mathop{\mathbb{E}}_{z\sim p(z)}\exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z,x,y)\right]-\ell^\theta(z,x_i,y_i)\right)\right)\\
&\quad=\prod_{i=1}^{n}\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim p(z)}\exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi}\mathop{\mathbb{E}}_{y'\sim\zeta}\left[\ell^\theta(z,x',y')\right]-\ell^\theta(z,x,y)\right)\right)\\
&\quad=\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim p(z)}\exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi}\mathop{\mathbb{E}}_{y'\sim\zeta}\left[\ell^\theta(z,x',y')\right]-\ell^\theta(z,x,y)\right)\right)\right)^n.
\end{aligned}$$

After applying a logarithm, we get

$$\log \mathop{\mathbb{E}}_{S_x \overset{iid}{\sim} \pi} \mathop{\mathbb{E}}_{S_y \overset{iid}{\sim} \zeta} \mathop{\mathbb{E}}_{Z \overset{iid}{\sim} p(z)} \exp\left(\frac{\lambda}{n} \sum_{i=1}^{n} \left(\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \left[\ell^\theta(z_i, x, y)\right] - \ell^\theta(z_i, x_i, y_i)\right)\right)$$

$$= \log \left(\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim p(z)} \exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi} \mathop{\mathbb{E}}_{y'\sim\zeta} \left[\ell^\theta(z, x', y')\right] - \ell^\theta(z, x, y)\right)\right)\right)^n.$$

$$= n \log \left(\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim p(z)} \exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi} \mathop{\mathbb{E}}_{y'\sim\zeta} \left[\ell^\theta(z, x', y')\right] - \ell^\theta(z, x, y)\right)\right)\right) \tag{6.20}$$

Combining (6.18) with the logarithmic results from (6.19) and (6.20) yields

$$\frac{\lambda}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)} \left[\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \left[\ell^\theta(z, x, y)\right] - \ell^\theta(z, x_i, y_i)\right] - \sum_{i=1}^{n} D_{KL}(q_\phi(z|x_i, y_i)\|p(z)) \leq$$

$$\log \frac{1}{\delta} + n \log \left(\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim p(z)} \exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi} \mathop{\mathbb{E}}_{y'\sim\zeta} \left[\ell^\theta(z, x', y')\right] - \ell^\theta(z, x, y)\right)\right)\right). \tag{6.21}$$

Dividing by λ and splitting up the terms on the left side yields

$$\frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)} \mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \left[\ell^\theta(z, x, y)\right] - \frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)} \left[\ell^\theta(z, x_i, y_i)\right]$$

$$- \frac{1}{\lambda} \sum_{i=1}^{n} D_{KL}(q_\phi(z|x_i, y_i)\|p(z)) \leq \frac{1}{\lambda}\Bigg[\log \frac{1}{\delta} \tag{6.22}$$

$$+ n \log \left(\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim p(z)} \exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi} \mathop{\mathbb{E}}_{y'\sim\zeta} \left[\ell^\theta(z, x', y')\right] - \ell^\theta(z, x, y)\right)\right)\right)\Bigg].$$

Rearranging the terms yields

$$\frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)} \mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \left[\ell^\theta(z, x, y)\right] \leq \frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)} \left[\ell^\theta(z, x_i, y_i)\right]$$

$$+ \frac{1}{\lambda}\Bigg[\sum_{i=1}^{n} D_{KL}(q_\phi(z|x_i, y_i)\|p(z)) + \log \frac{1}{\delta} \tag{6.23}$$

$$+ n \log \left(\mathop{\mathbb{E}}_{x\sim\pi} \mathop{\mathbb{E}}_{y\sim\zeta} \mathop{\mathbb{E}}_{z\sim p(z)} \exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi} \mathop{\mathbb{E}}_{y'\sim\zeta} \left[\ell^\theta(z, x', y')\right] - \ell^\theta(z, x, y)\right)\right)\right)\Bigg].$$

Using the definition of an IMP and Lemma 6.1 for any $x_i \in S_x, y_i \in S_y, x \in \mathcal{X}, y \in \mathcal{Y}$, we have

$$\begin{aligned}\mathop{\mathbb{E}}_{z\sim q(z|x,y)}\left[\ell^\theta(z,x,y)\right] - \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\ell^\theta(z,x,y)\right] &\leq d_{\mathcal{E}}(q_\phi(z|x,y), q_\phi(z|x_i,y_i)) \\ &\leq K_\phi K_\theta\left(d(x,x_i)+d(y,y_i)\right)+K_\theta d(x,x_i),\end{aligned} \tag{6.24}$$

where $d(y,y')=\|y-y'\|$. Applying Fubini's theorem yields

$$\begin{aligned}\sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim q(z|x,y)}\left[\ell^\theta(z,x,y)\right] - \sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\ell^\theta(z,x,y)\right] \\ \leq \sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[K_\phi K_\theta\left(d(x,x_i)+d(y,y_i)\right)+K_\theta d(x,x_i)\right].\end{aligned} \tag{6.25}$$

Rearranging the terms yields

$$\begin{aligned}&\sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\ell^\theta(z,x,y)\right] \\ &\quad\geq \sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\mathop{\mathbb{E}}_{z\sim q(z|x,y)}\left[\ell^\theta(z,x,y)\right] - K_\phi K_\theta\left(d(x,x_i)+d(y,y_i)\right) - K_\theta d(x,x_i)\right].\end{aligned} \tag{6.26}$$

We note that

$$\sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\ell^\theta(z,x,y)\right] = \sum_{i=1}^n \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\ell^\theta(z,x,y)\right], \tag{6.27}$$

so combining (6.26) with (6.23) yields

$$\begin{aligned}&\frac{1}{n}\sum_{i=1}^n \mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\left[\mathop{\mathbb{E}}_{z\sim q(z|x,y)}\left[\ell^\theta(z,x,y)\right] - K_\phi K_\theta\left(d(x,x_i)+d(y,y_i)\right) - K_\theta d(x,x_i)\right] \\ &\quad\leq \frac{1}{n}\sum_{i=1}^n \mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\left[\ell^\theta(z,x_i,y_i)\right] + \frac{1}{\lambda}\Bigg[\sum_{i=1}^n D_{KL}(q_\phi(z|x_i,y_i)\|p(z)) + \log\frac{1}{\delta} \\ &\qquad + n\log\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim p(z)}\exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi}\mathop{\mathbb{E}}_{y'\sim\zeta}\left[\ell^\theta(z,x',y')\right] - \ell^\theta(z,x,y)\right)\right)\right)\Bigg].\end{aligned} \tag{6.28}$$

Finally, rearranging terms yields

$$\begin{aligned}\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim q(z|x,y)}\Big[\ell^\theta(z,x,y)\Big] \leq \frac{1}{n}\sum_{i=1}^{n}\mathop{\mathbb{E}}_{z\sim q(z|x_i,y_i)}\Big[\ell^\theta(z,x_i,y_i)\Big] \\ + \frac{1}{\lambda}\Bigg[\sum_{i=1}^{n} D_{KL}(q_\phi(z|x_i,y_i)\|p(z)) + \log\frac{1}{\delta} \\ + \frac{\lambda K_\phi K_\theta}{n}\sum_{i=1}^{n}\left(\mathop{\mathbb{E}}_{x\sim\pi}[d(x,x_i)] + \mathop{\mathbb{E}}_{y\sim\zeta}[d(y,y_i)]\right) + \frac{\lambda K_\theta}{n}\sum_{i=1}^{n}\mathop{\mathbb{E}}_{x\sim\pi}[d(x,x_i)] \\ + n\log\left(\mathop{\mathbb{E}}_{x\sim\pi}\mathop{\mathbb{E}}_{y\sim\zeta}\mathop{\mathbb{E}}_{z\sim p(z)}\exp\left(\frac{\lambda}{n}\left(\mathop{\mathbb{E}}_{x'\sim\pi}\mathop{\mathbb{E}}_{y'\sim\zeta}\Big[\ell^\theta(z,x',y')\Big] - \ell^\theta(z,x,y)\right)\right)\right)\Bigg].\end{aligned} \tag{6.29}$$

□

Similar to the CVAE objective, a crucial assumption underlying the concept of PAC-Bayes bounds is the ability to draw *i.i.d.* samples. Ideally, an agent (or algorithm) would exhaustively sample all regions of the data distribution simultaneously and produce a function from the ensemble of data samples. This is equivalent to offline learning, where the learning process has access to a complete dataset before training. When we instead consider an embodied learning process, an agent must navigate the exploration domain to gather samples in a time-ordered sequential sampling process. In general, such a sampling process does not produce *i.i.d.* data [**180**]. However, ergodicity can provide a method of producing *i.i.d.* data with Birkhoff's ergodic theorem [**189**], which we restate below:

Theorem 6.2 (Birkhoff's Ergodic Theorem) *Let $\{x_t\}_{t\in N}$ be an aperiodic and irreducible Markov process on a state space X with invariant measure ρ and let $f : \mathcal{X} \to \mathbb{R}$ be any measurable function with $\mathbb{E}[|f(x)|] < \infty$. Then,*

$$\lim_{N\to\infty}\frac{1}{N}\sum_{t=1}^{N} f(x_t) = E_{x\sim\rho}[f(x_0)] \tag{6.30}$$

almost surely.

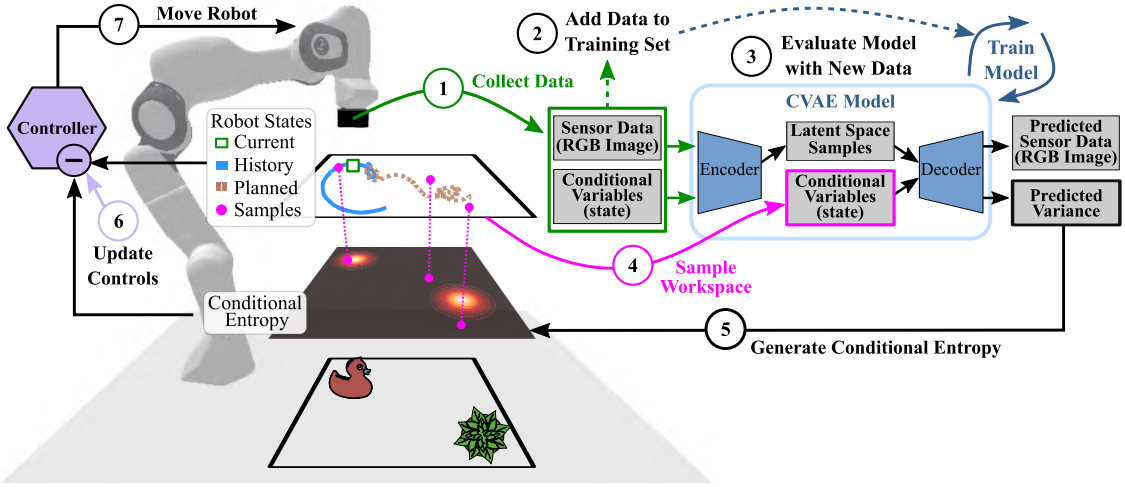

Figure 6–2 Closed-Loop Active Learning Process Numerals 1-7 describe the steps completed during each active learning step. The layers below the robot arm show, (I) previously visited states, future planned states, and samples drawn from the reachable workspace (6.31), (II) conditional entropy of the model over the workspace samples (6.32), and (III) objects in the workspace. Model training occurs in parallel to data collection, so the robot trajectory, model, dataset, and conditional entropy all evolve continually.

Informally, this theorem states that the time average of any function of an ergodic Markov chain is equal to its ensemble average. We can use ergodic control strategies from [**144**, **146**] to generate data with *i.i.d.* properties asymptotically. In the next section, we describe our active learning approach that leverages an ergodic control strategy.

6.2 Methods

With embodied learning, how informative the model is depends on what data the robot has collected. Our experimental pipeline contains three steps—active learning, object fingerprinting, and object identification (see Fig. 6–1). Our active learning process couples the robot data collection process (*i.e.*, exploration) with the CVAE model learning process. The numeral bubbles in Fig. 6–2 describe the process completed during each robot exploration step—1) collect data, 2) add data to training set, 3) evaluate the CVAE model for

latest collected data, 4) sample workspace, 5) generate conditional entropy distribution using predicted decoder variance evaluated at each workspace sample, 6) update controller to minimize the difference between visited states and conditional entropy, and 7) move robot to new state. The robot repeats steps 1-7 until learning is complete. Model training occurs in parallel to data collection, so the robot trajectory, model, and conditional entropy are all continually evolving. We outline the active learning process in Algorithm 6–1. See Appendix C for additional implementation details including the model architecture.

6.2.1 Control Formulation

In this work, we use a receding-horizon ergodic controller from [**144**] to synthesize future controls. As was introduced in Section 2.4.1, the exploration strategy seeks to minimize the KL-divergence between the target spatial distribution h, and the states visited by the robot in the form of a time-average statistics d. We define these distributions in terms of s, which are sample states drawn from the reachable workspace $\mathcal{X}$. We define the time-average statistics of the robot as

$$d(s|x(t)) = \frac{1}{T_r}\int_{t_i-T_r}^{t_i+T} \frac{1}{\eta}\psi(s|x(t))dt \tag{6.31}$$

where $\psi(s|x(t)) = \exp(-\frac{1}{2}\|s - x(t)\|^2_{\Sigma^{-1}})$, η is a normalization factor, T is the planning horizon, and Σ specifies the width of the state coverage Gaussian.

We use the method from [**15**] to specify the target spatial distribution h, which incorporates model uncertainty into the ergodic control strategy by specifying the target distribution as the conditional entropy distribution of model uncertainty over the possible search space.

We define the CVAE conditional entropy distribution as

$$\begin{aligned} h(s|x,y) &= \exp\left(\mathbb{H}\Big[p_\theta\left(y|z_c,s\right)\Big]\right)^k \\ &\propto \exp\left(\frac{1}{2}\ln\left(\mathrm{Var}\Big[p_\theta\left(y|z_c,s\right)\Big]\right)\right)^k \end{aligned} \tag{6.32}$$

where $z_c \sim q_\phi(z|x_c, y_c)$ is a latent space conditioned on the current robot state and sensor data, $\mathbb{H}$ is the entropy of the decoder network p_θ conditioned on z_c, and k is an effective temperature parameter that exponentially weighs regions of high importance. We overload z with the subscript c to highlight that the distribution is continually changing in response to the latest collected robot data. One way to interpret this conditional entropy distribution is as the expected information density. Areas with high conditional entropy are *very interesting* and areas of low conditional entropy are *not very interesting*.

Since our embodied system collects training data from scratch, it may take time from the robot to collect data throughout the training workspace–particularly as the size of the workspace grows, the state dimension increases, or the speed of the robot decreases. We introduce workspace coverage as a measure of the amount of the reachable workspace that the robot has visited. When the workspace coverage is low, we want to uniformly cover the workspace to ensure all possible objects are discovered, but as coverage increases, we want our controller to focus on matching the conditional entropy. To incorporate workspace coverage into the target distribution, we dynamically tune the temperature hyperparameter from (6.32), such that

$$k = \frac{\eta}{S}\sum_{i=1}^{S}\max_{x\in\mathcal{D}}(\psi(s_i|x)) \tag{6.33}$$

where $\mathcal{D}$ is the set of all previously visited states, s are samples drawn from the reachable workspace, ψ is defined in (6.31), and η is an additional scaling term.[1] k increases as the collected data covers the reachable workspace.

By collecting data from the environment that matches the conditional entropy distribution, we satisfy the *i.i.d.* requirements of the CVAE learning bounds. Generative models like CVAEs are often criticized for *hallucinations* (*i.e.*, imagining features where in reality there are no features). But hallucinations are only bad if you can't collect data to confirm or deny if the *imagined* features are real. With active learning, our embodied robot can collect data from uncertain regions (both real and imagined). Although the conditional distribution evolves as data is collected, the true underlying distribution remains constant and the algorithm asymptotically approaches the true distribution. We use the KL-ergodic measure $D_{KL}(h\|d)$ to assess the data collection quality over time.

6.2.2 Embodied CVAE Loss Formulation

To satisfy CVAE PAC-Bayes Bounds from Theorem 6.1, we augment the CVAE loss function from (2.4) with an additional conditional term such that

$$\begin{aligned}\mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y}) = \underbrace{\mathbb{E}_{q_\phi(z|x,y)}[\log p_\theta(y|z,x)]}_{\text{end-to-end reconstruction loss}} - \beta \underbrace{D_{KL}(q_\phi(z|x,y) \parallel p(z))}_{\text{latent space regularization}} \\ + \gamma \underbrace{\mathbb{E}_{q_\phi(z|x,y)}[\log p_\theta(y'|z,x')]}_{\text{conditional reconstruction loss}}\end{aligned} \tag{6.34}$$

where β and γ are hyperparameters Both the end-to-end reconstruction loss and the conditional reconstruction loss use the same encoded latent space, but they use different conditional states, which are indicated with the prime notation.

[1] Different values for the width of the state coverage Gaussian (Σ. Until the ultrasound experiments, all tests use $\eta = 1$) or different numbers of workspace samples S could be chosen for $d(s)$ and k if a coarser or finer coverage metric is desired.

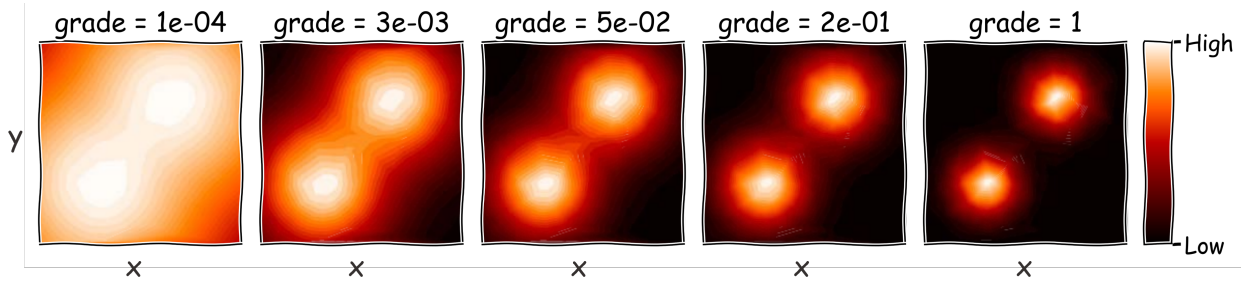

Figure 6–3 Conditional Entropy Grade Illustration. Each heatmap represents a conditional entropy distribution with high values shown in white and low values shown in black. The title shows the grade calculated for each conditional entropy map. As the peaks get taller, the area of the light colored regions gets smaller, and the grade increases.

We use the conditional entropy distribution from (6.32) to dynamically tune the latent space regularization hyperparameter from (6.34). If we think of the conditional entropy map as a landscape, the *peaks* in the landscape represent objects in the environment. As the model learns to differentiate objects, the *peaks* in the conditional entropy distribution get *taller*. To tune β according to the steepness of the conditional entropy *peaks*, we introduce the *grade* of the conditional entropy as

$$grade = \exp_{10}\left(-\log_{10}\left(\min\frac{h(s)}{\max h(s)}\right)-\chi\right), \tag{6.35}$$

where χ is the scale of the minimum desired grade (*e.g.*, $\chi = 4$ corresponds to $\min m = 10^{-4}$). By setting β equal to the grade, β increases as the model learns to discriminate objects.[2] See Fig. 6–3 for an illustration of grades for different conditional entropy distributions. We use the workspace coverage from (6.33) to dynamically tune the reconstruction hyperparameter from (6.34) with $\gamma = 0.1k$. Over time, γ increases as more data is collected from the search space. We assess the impact of these hyperparameters in Section 6.4.

[2] We define grade as m rather than directly specifying β to allow us to assess this metric during ablation studies with fixed β later in this chapter.

Algorithm 6–1 Active Learning Process

Randomly initialize CVAE encoder $q_\phi(z|x,y)$ and decoder $p_\theta(y|z,x)$. Initialize memory buffer $\mathcal{D}$, prediction horizon H, state coverage Σ, and latent space regularization scale χ.
while task not done **do** ——— **Model Training**
  Collect current state x_c and sensor data y_c and add to buffer $\mathcal{D} \leftarrow \mathcal{D} + \{x_c, y_c\}$
  Generate latent space samples for current data $z_c \sim q_\phi\left(z|x_c, y_c\right)$
  Draw S states from reachable workspace
  Draw N previous states from buffer and predict future states over horizon H
  Update distributions over workspace samples $(s_1, \ldots, s_n)$
    Workspace Coverage: $k \leftarrow \frac{1}{S}\sum_{i=1}^{S} \max_{x\in\mathcal{D}} \left(\exp\left(-\frac{1}{2}\|s_i - x\|^2_{\Sigma^{-1}}\right)\right)$
    Conditional Entropy Distribution: $h(s) \leftarrow \exp\left(\mathbb{H}(p_\theta\left(y|z_c, s\right))\right)^k$
    Trajectory Distribution: $d(s) \leftarrow \sum_{t=1}^{N+H} \exp\left(-\frac{1}{2}\|s - x_t\|^2_{\Sigma^{-1}}\right)$
  Update controller by minimizing $D_{KL}(h(s) \parallel d(s))$
  Update CVAE hyperparameters [3]
    $\beta \leftarrow \exp_{10}\left(-\log_{10}\left(\min\left(h/\max h\right)\right) - \chi\right)$
    $\gamma \leftarrow 0.1k$
  Update CVAE parameters ϕ, θ with mini-batches from $\mathcal{D}$ [3]
  Apply next action from controller to move robot
end while
Store CVAE encoder $q_\phi(z|x,y)$ and decoder $p_\theta(y|z,x)$ for future tasks
if fingerprint **then** ——— **Object Fingerprinting**
  Cluster $h(s)$ into K objects [4]
  for $k \in \{1, \ldots, K\}$ **do**
    Move robot to object center x_k and collect sensor data y_k
    Initialize fingerprint buffer $\mathcal{F}_k$ and data collection method
    while fingerprint buffer not full **do**
      Move robot to next fingerprint state
      Collect current state x_c and sensor data y_c
      Generate latent space distribution for current data $z_c = q_\phi\left(z|x_c, y_c\right)$
      Add data to buffer $\mathcal{F}_k \leftarrow \mathcal{F}_k + \{x_c, z_c\}$
    end while
    Store fingerprint data $\{x_k, y_k, \mathcal{F}_k\}$ for future tasks
  end for
end if

[3] CVAE updates can also occur asynchronously with distributed training.
[4] Clustering can also be performed during model-training to assess the stability of detected objects.

6.2.3 Active Units

An aim of this work is to actively learn latent features of unknown objects. In Section 2.2, we introduced latent space posterior collapse and a method to evaluate the number of latent space active units from [**68**]. We restate the active units equation here for section readability.

$$\text{AU} = \sum_{l=1}^{L} \mathbf{1}[\text{Var}_{\mathcal{S}}(\mu_{(z,l)}) > \tau], \tag{2.3}$$

where L is the size of the latent space, μ_z are the latent space means, $\mathcal{S}$ a subset of the collected data points, $\text{Var}(\cdot)$ is the variance across $\mathcal{S}$, $\mathbf{1}[\cdot]$ is an indicator function, and τ is the active unit threshold. When a latent space fully collapses, the latent space contains no active units. The larger the number of active units, the more latent space dimensions are being used by the decoder.

6.2.4 Object Fingerprinting

The robot identifies learned objects in the environment by clustering samples from the conditional entropy distribution. Clustering can also be used during training to assess the model's belief about the number of objects at any given time. We refer to *fingerprinting* as the act of collecting data to represent these objects for future identification tasks. We use the kernel-based ergodic metric from [**214**] to generate samples matching the conditional entropy distribution. We use Mean Shift Clustering to group these samples into object locations [**215**]. Since objects are defined relative to the workspace, "blank" regions are valid object locations if they are different from other workspace regions. For each object, the robot generates a lookup table of states and latent space variables (*i.e.*, what we call a fingerprint). Along with the learned model, these fingerprints can be used for object

identification in new environments. We outline the fingerprinting process in the second half of Algorithm 6–1.

Algorithm 6–2 Object Identification

Load CVAE encoder $q_\phi(z|x,y)$ and decoder $p_\theta(y|z,x)$. Load fingerprint(s) $\{x_k, y_k, z_k, \mathcal{F}_k\}$. Initialize belief grid(s) G_k and measurement model(s). Initialize test buffer $\mathcal{D}$. Initialize data collection method.[5]
while task not done **do**
    Collect state x_c and sensor data y_c
    for $k \in \{1, \ldots, K\}$ **do** ▷ Test Fingerprints
        Empty test buffer $\mathcal{D}$
        for $(x_s, y_s, z_s) \in \mathcal{F}_k$ **do** ▷ Test each stored fingerprint state
            Generate latent space distribution $z_t = q_\phi\left(z|x_s, y_t\right)$
            Calculate distance $m_s = \|z_s - z_t\|$
            Add data to buffer $\mathcal{D} \leftarrow \mathcal{D} + \{x_s, m_s\}$
        end for
        Find most likely measurement $(x^*, m^*) = \arg\min_m(\mathcal{D})$
        Generate likelihood using (x_c, x_k, x^*, m^*) and measurement model ▷ Get Belief
        Update belief grid G_k ▷ Bayesian Update
    end for
    Apply action to move robot
end while Store belief grids

[5] When active search is used, we initialize a new CVAE model within the data collection system.

6.2.5 Object Identification

During object identification, the learned CVAE model and object fingerprints are used as measurement models to test data collected from the new environment. For each identification test environment, a new model is learned from scratch using the active learning method from Algorithm 6–1. Data collected by the new model is processed by each learned measurement model and incorporated into belief grids for each learned object fingerprint. Each belief grid-point has a mean and a standard deviation representing how likely it is that a particular object is present in the test workspace. The following are completed for each test image: 1) the stored CVAE encoder generates latent space samples for collected image paired with

each stored fingerprint state; 2) the distance[6] between stored latent space and each new latent space is calculated; 3) a likelihood grid is generated using the transform between test environment and most likely measurement; 4) the belief grid is updated using Bayes rule. For these tests, we use the minimum distance between fingerprints as the distance threshold for the measurement model. We outline the identification process in Algorithm 6–2.

6.3 RGB Camera Experiments

The experiments in this section were designed to highlight the ability of our method to actively learn information-rich latent spaces in an embodied framework. For the examples in this section, restrict our robot to move in the planar workspace (x, y, θ) to collect data with an RGB camera (see Fig. 6–1). The robot has no pre-specified representation of camera properties or the objects in the workspace. First, we demonstrate active learning with the original CVAE loss function. Then, we demonstrate embodied active learning with our new, decoder-conditioned CVAE loss function.

6.3.1 Active Learning (Original CVAE Loss)

For our first set of experiments, we train CVAE models with three different view selection methods—1) active coverage, 2) random walk, and 3) uniform data distribution. In this section, we use the original loss function[7] from (2.4) and ramp the β term linearly from 0 to 0.05 over 10,000 model updates. Our active coverage method uses the process described in Algorithm 6–1. The random walk method uses random sampling in the action space.

[6] We typically use the L2 norm as our distance measurement, but alternate options like the KL-divergence are also possible.

[7] The original loss function in (2.4) is equivalent to (6.34) with $\gamma = 0$.

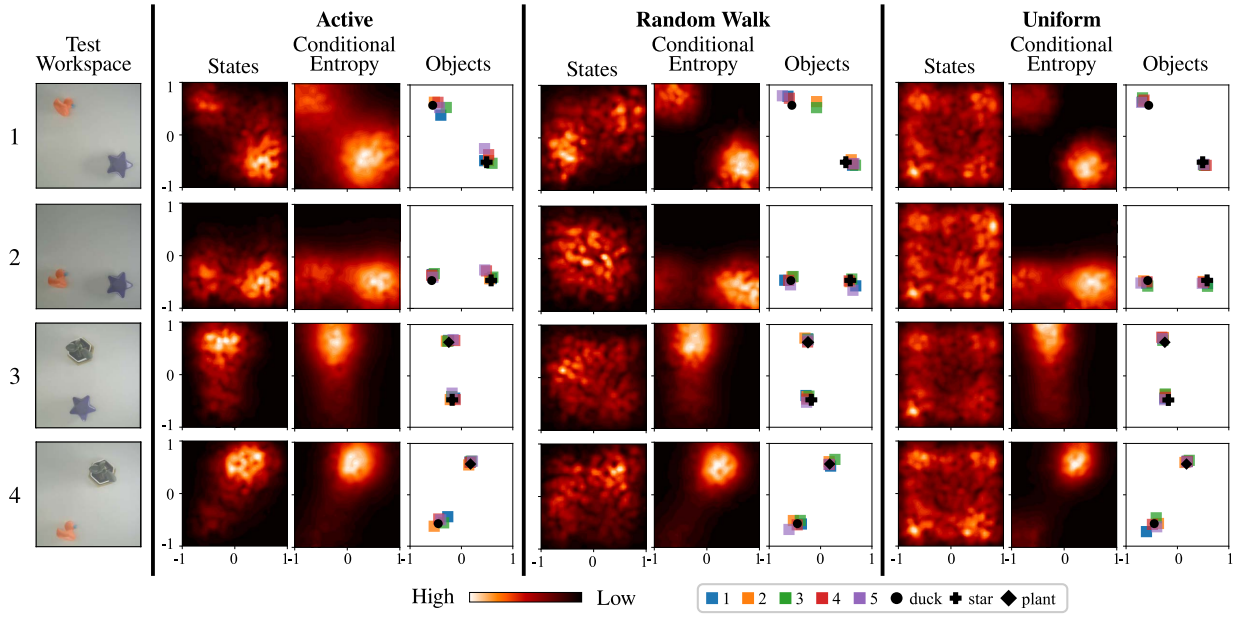

Figure 6–4 RGB Camera CVAE Learning Distributions. Each row shows the results for a different test workspace. The first column shows a top-down view of the workspace. The other plots summarize results for three different data collection methods after training the CVAE model for 3000 steps. Heatmaps show representative examples for one seed, where *high* values are white and *low* values are black. *States* heatmaps show the distribution of states visited during the test. *Conditional Entropy* heatmaps show the uncertainty belief of the model. The object scatter plots show the object location extracted from the conditional entropy maps; each color shows a different seed and each black marker shows a different object. These plots show that all methods generate conditional entropy maps with high regions matching the true object locations, but the random walk and uniform methods do not incorporate this information into their exploration strategy, as represented by the state distributions. Moreover, the conditional entropy maps indicate that the plant is substantially more interesting than the star or duck, capturing its feature rich representation.

The uniform data distribution method chooses the next actions to minimize the difference between the new data and the previously collected data in terms of workspace states. For these experiments, we allow the robot to explore for 3000 time steps. We ran five different seeds (model initializations) for each test workspace. Fig. 6–4 shows each test workspace as well as representative conditional entropy and state distributions. These plots show that the states visited by our active method are consistent with the conditional entropy. Furthermore, the "high" regions of the conditional entropy distributions map onto the locations of objects

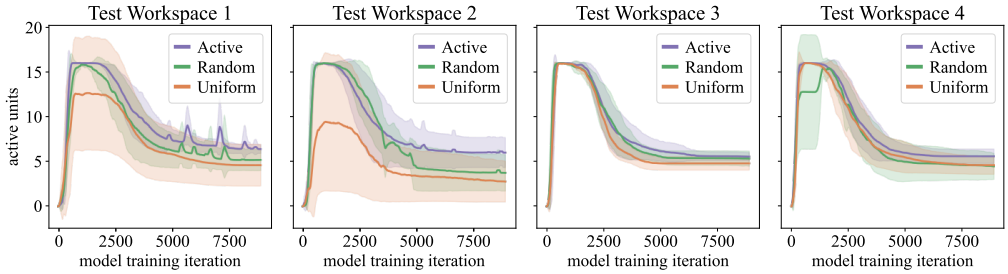

Figure 6–5 RGB Camera Latent Space Activity. Each subplot shows results for a different test workspace from Fig. 6–4. The solid lines show the average latent space activity across model training iterations and the shaded region shows the standard deviation across five seeds.

	Active		**Random Walk**		**Uniform**	
Tests	Mean AU	Failure Rate	Mean AU	Failure Rate	Mean AU	Failure Rate
1	**6.0**	**0 / 5**	4.2	1 / 5	2.8	2 / 5
2	**6.4**	**0 / 5**	5.2	0 / 5	4.6	1 / 5
3	**5.6**	**0 / 5**	4.4	1 / 5	4.6	0 / 5
4	**5.4**	**0 / 5**	5.2	0 / 5	2.8	0 / 5

Table 6–1 RGB Camera Latent Space Activity The four test workspaces are shown in Fig. 6–4. Active Units (AU) is described in (2.3). Failed tests have AU = 0.

in the workspace. The conditional entropy maps indicate that the plants are more interesting than the plastic ducks and stars, reflecting the larger number of features required to learn the plant.

In Section 6.1, we discussed latent space posterior collapse. When a latent space fully collapses, the latent space contains no active units, and we consider that a failed test. Fig. 6–5 shows the latent space activity over time during model training, where we use a mini-batch of samples drawn from the full collected data. For all tested workspaces, our active method has more latent space activity and less variance across that activity than the other two exploration methods—this active learning approach is better for learning latent representations. Table 6–1 shows the final latent space unit activity averaged across all seeds and the percent of failed tests for each method. For this table, we use the entire data set collected during training. The results show our active method successfully learned models for all tested seeds

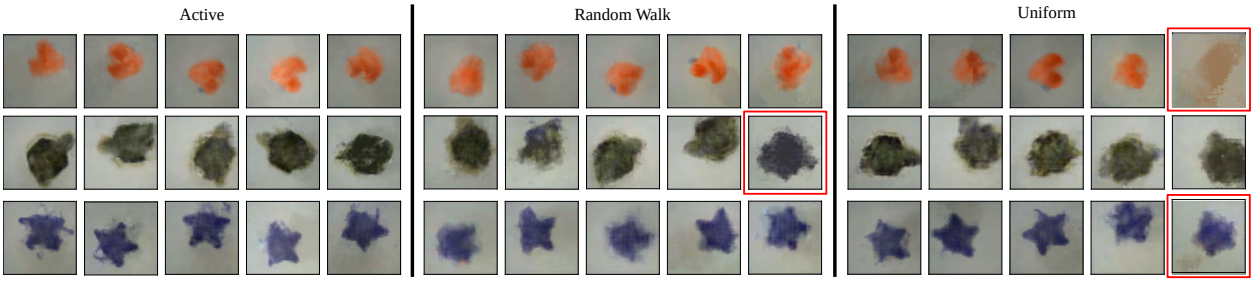

Figure 6–6 Representative reconstructions. The images show representative reconstructions for each exploration method. Images were reconstructed from the data sets collected for each test, so the exact position and orientation varies between figures. The columns are drawn from different seeds and the rows show different objects. The red boxes show instances where the latent space fully collapsed. Latent space collapse means that the decoder is ignoring the latent space, but because the decoder also incorporates the conditional variables, the model can still produce good state-based image predictions. If a learning goal is to produce an information-rich latent space, solely examining image reconstructions is not enough to determine the quality of the latent space.

in all test cases. Furthermore, our active method has more active units than random walk and uniform for all test cases.

Representative reconstructed images are shown in Fig. 6–6. Our active method produces similar quality reconstructions across all seeds. For the random walk and uniform exploration methods, there is a wider range of image quality—although some seeds produce crisp images, others are blurry. Instances with full latent space collapse are indicated with red boxes. These images look similar to those with information-rich latent spaces, highlighting the need for additional quality measures—like the number of active units—if the goal is to use the latent space in the future.

6.3.2 Object Identification (Original CVAE Loss)

For our next set of experiments, we focus on using our active exploration method for identification. After model training, four fingerprint locations were extracted from each CVAE's conditional entropy distribution. In Fig. 6–7, the top row shows the test workspace

and the fingerprint locations on the conditional entropy distribution. Fig. 6–7 also shows representative CVAE outputs for each fingerprint location. Images from the cluster means were processed by the CVAE encoder and the resulting mean latent space values are plotted. The remaining results in this section only use the actively learned CVAE because the random CVAE did not produce usable latent space variables. Since two of the fingerprints were blank, we excluded the redundant location from object identification. We tested the same actively learned models and fingerprints in three test configurations: 1) the original learning environment 2) a test environment with an additional plant 3) a test environment with additional ducks. The number of objects in each test configuration is not provided to the active object identification module. The figures in this section are all for one seed; see Appendix C for 9 other seeds.

For each actively learned belief figure, the left side contains the belief grids for fingerprint identification in the learning environment. One belief update was incorporated for each image collected in the test environment (*i.e.*, one image per step). To display 3D data in 2D figures, these grids represent x-y positions averaged across all angles. Black regions indicate areas of low-likelihood and white regions indicate areas of high-likelihood. The figure also shows additional information for each "high-likelihood" region on the duck and plant grids—actual images, imagined images, and angle belief. The "actual" images are taken by the robot in the test environment at the x-y position indicated by the tail of the arrows and the image angle indicated by a green square in the far right column. The "imagined" image is the output from the CVAE predicted by inputs of the "actual" image and fingerprint center state.

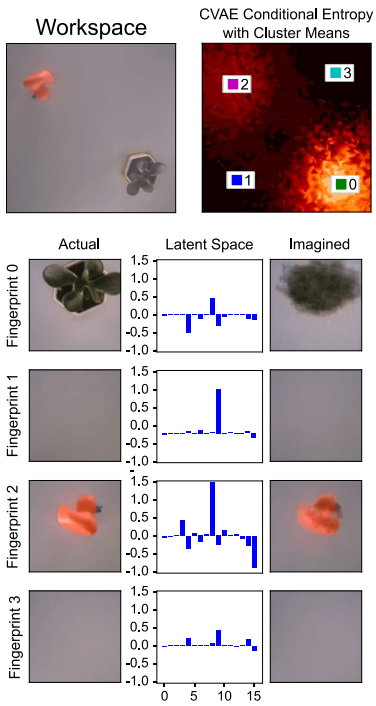

Figure 6–7 Actively Learned Fingerprints. The top row shows the test workspace and the locations of the learned cluster means (fingerprint locations). For each fingerprint, a real image, latent space (encoder output), and imagined image (decoder output) is included. Qualitatively, the rows from top to bottom can be identified as *plant*, *blank*, *duck*, and *blank* respectively, but the algorithm has no knowledge of these labels. These fingerprints show the model learned 1) a latent space representation and 2) an image reconstruction of the imagined data.

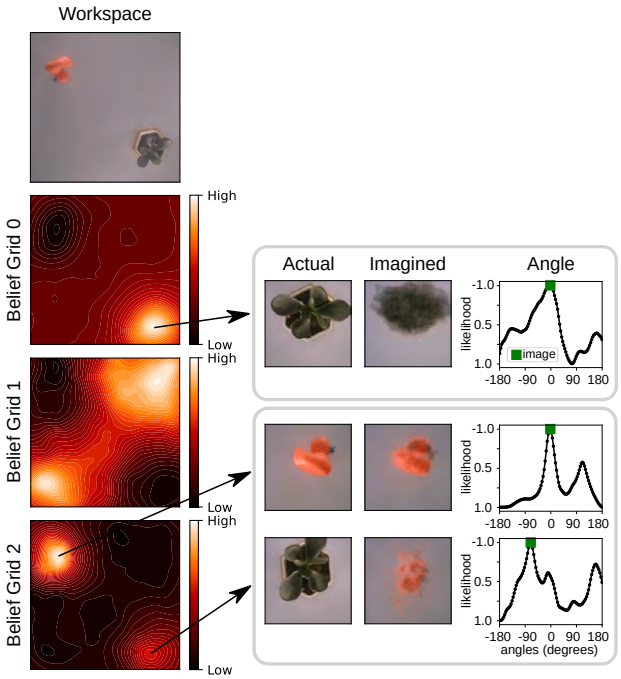

Figure 6–8 Actively Learned Belief in Training Environment. The first column shows belief grids for each object with high-likelihood regions in white and low-likelihood regions in black. See Fig. 6–7 for learned object locations. For Grids 0 and 2, pop-outs are provided for each high-likelihood region with additional information at that x-y location—actual image, imagined image, and normalized angle likelihoods.

Learning Environment. Fig. 6–8 summarizes the actively learned fingerprint identification information in the learning environment after 1000 time steps. Belief Grids 0, 1, and 2 correspond to *plant*, *blank*, and *duck* respectively. From Belief Grid 0, we can interpret that the *blank* regions are more "plant-like" than the *duck* object. From Belief Grid 1, we can interpret that the *blank* regions are distinct from both the *duck* and the *duck*. From Belief

Grid 2, we can interpret that the *plant* object is more "duck-like" than the *blank* regions. The differences between Grid 0 and Grid 2 are likely due to the symmetry differences between the two objects. The *duck* looks different from every orientation while the *plant* has symmetry and can look the same from several orientations. Furthermore, we would expect there to be similar features between both duck and plant objects, and the faint *plant* location in the *duck* belief demonstrates feature overlap. The belief grids are currently based only on the latent space produced by the CVAE encoder, but the imagined image and uncertainty produced by the decoder could be used to further disambiguate "true" ducks from "faint" ducks. Comparing the belief grids to the workspace and cluster centers qualitatively confirms correct identification.

Plant Environment. Fig. 6–9 summarizes the actively learned fingerprint identification information after 1000 time steps. In this case, the additional plant isn't exactly the same as the test plant. Belief Grid 0 correctly identifies the new object as more "plant-like" than "duck-like". Again, in Belief Grid 2, the *plants* are weakly identified as more "duck-like" than empty due to feature overlap between the *duck* and *plant* objects.

Duck Environment. Fig. 6–10 summarizes the actively learned fingerprint identification information after 3000 time steps. In this case, the objects are much closer together than in the previous tests. Belief Grid 0 correctly identifies all of the new objects as more "duck-like" than "plant-like" even though they are oriented at new, untrained orientations. In Belief Grid 2, the *ducks* are correctly identified as "duck-like".

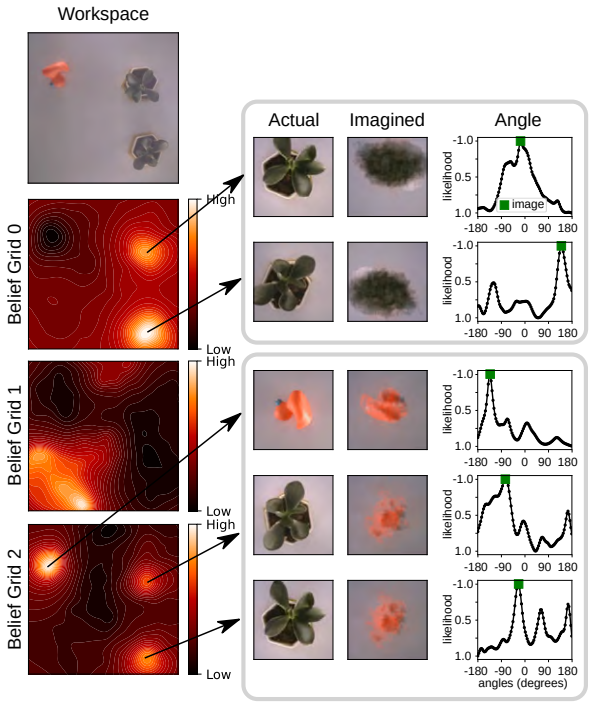

Figure 6–9 Actively Learned Belief in Plant Environment. The first column shows belief grids for each object with high-likelihood regions shown in white and low-likelihood regions in black. See Fig. 6–7 for learned object locations. For Grids 0 and 2, pop-outs are provided for each high-likelihood region with additional information at that x-y location—actual image, imagined image, and normalized angle likelihoods. Grid 0 indicates that the new object has a high-likelihood of being a plant, and Grid 2 indicates that the new object has a low-likelihood of being a duck.

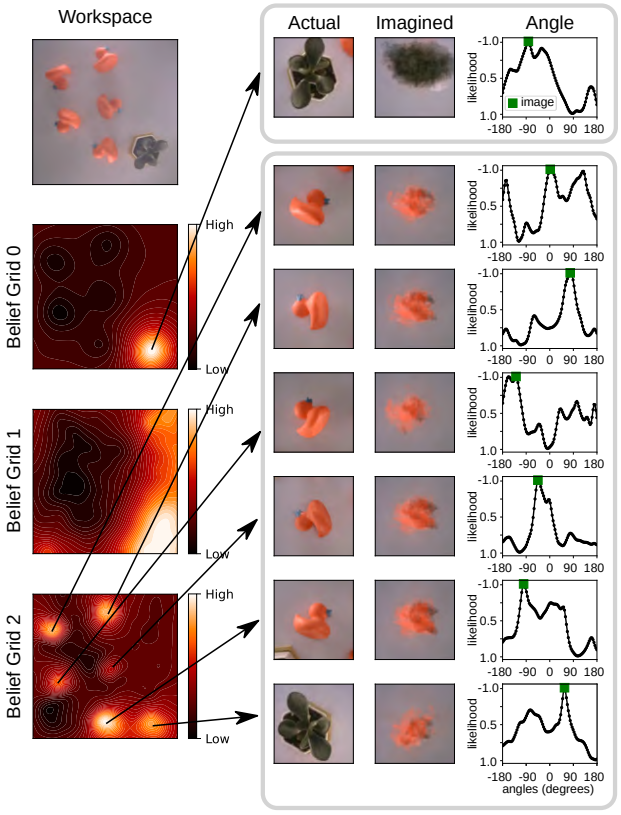

Figure 6–10 Actively Learned Belief in Duck Environment. The first column shows belief grids for each object with high-likelihood regions in white and low-likelihood regions in black. See Fig. 6–7 for learned object locations. For Grids 0 and 2, pop-outs are provided for each high-likelihood region with additional information at that x-y location—actual image, imagined image, and normalized angle likelihoods. Grid 0 indicates that the new objects have a low-likelihood of being plants, and Grid 2 indicates that the new objects have a high-likelihood of being ducks.

6.3.3 Adaptive, Active Learning (New Embodied CVAE Loss)

For our next set of experiments, we train CVAE models with our new loss function from (6.34), which includes the new decoder-conditioned term. We adaptively tune the hyperparameters (β, γ). Fig. 6–11 shows the learning metrics for four different training workspaces, each containing two objects. For each workspace, we test five algorithmic seeds (20 tests total). All active models learn information-rich latent spaces and quickly collect data consistent with their conditional entropy distributions. The last column of Fig. 6–11 shows data collected with a random walk, which collapses for 2/5 seeds. Comparing these active unit results to Fig. 6–5 shows that the new loss function allows the latent space to retain full activation across all seeds and test workspaces.

After training, we generate fingerprints for each actively learned object. For the results in this section, we collected 50 data samples per object. We first test these fingerprints against their respective training environments. Then, we test these fingerprints in new workspaces with extra objects shown in Fig. 6–12. For comparison purposes, we collect 2000 test images for each identification test and test all 60 collected fingerprints on the same images.[8] Therefore, the data quality of the identification process is solely the result of objects in the environment, not the belief about the presence of particular fingerprints.

[8] We get 60 fingerprints from 4 test environments $\times$ 5 seeds per environment $\times$ 3 fingerprints per environment

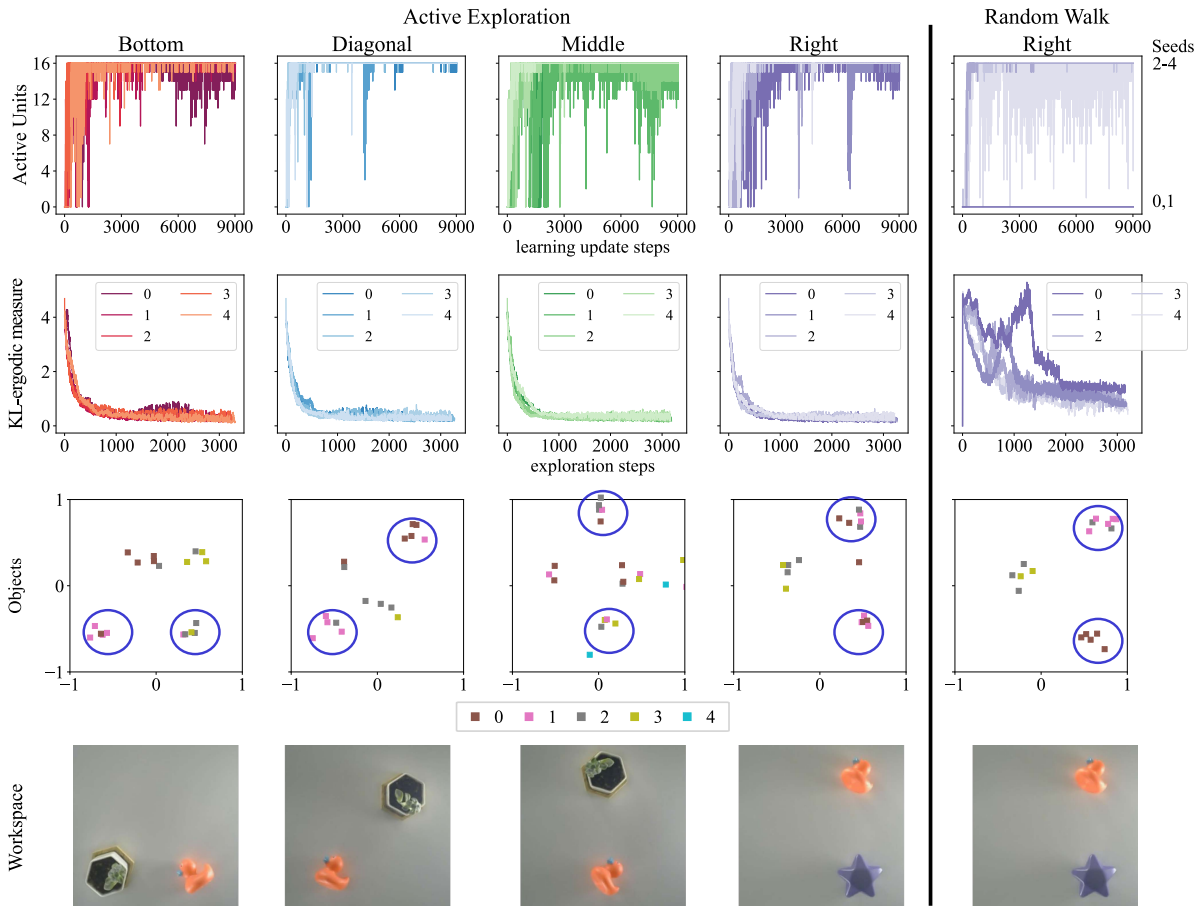

Figure 6–11 Learning Metrics. The first four columns show active exploration for different learning environments with 5 different seeds. The last column shows data collected with a random walk. The top row shows the number of active units for each mini-batch update. Zero indicates latent space collapse, and the higher the number of active units, the more the latent space is being used by the decoder. The second row shows the ergodic metric for each exploration step—representing how well the collected data matches the conditional entropy distribution. An ergodic measure of zero would mean the collected data exactly matches the target distribution. The third row shows identified object locations relative to the x-y workspace; blue circles indicate the ground truth locations of objects in the workspace. Different methods found different numbers of "blank" objects depending on the particular conditional entropy distribution, but all methods found objects located at the true object locations in the workspace. The bottom row shows a top-down view of each learning workspace. These plots show for all active seeds, the number of active units remained high and collected data quickly converged to the conditional entropy distribution. For the random walk, two seeds experienced latent space collapse and the data collected did not match the conditional entropy distribution. For all seeds, the new conditional loss term enables the random walk to identify regions of uncertainty, although it is unable to act on them.

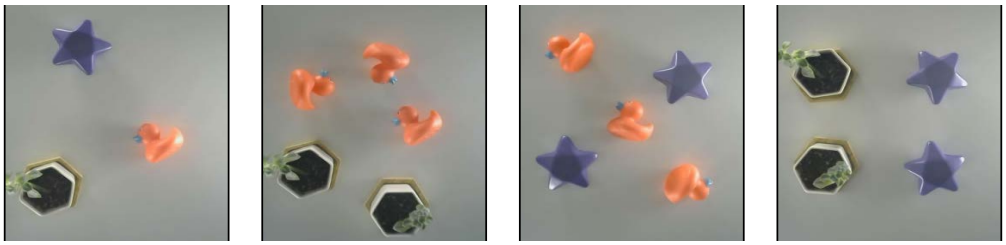

Figure 6–12 Test Workspaces

	Objects					Objects			*Total by*
Training	Duck	Plant	Star	Blank	**Test**	Duck	Plant	Star	***Test Env.***
Bottom	5/5	5/5	0/0	5/5	Star,Plant,Duck	18/20	14/15	4/5	*90%*
Diagonal	5/5	5/5	0/0	5/5	Plants & Ducks	54/60	28/30	0/0	*92%*
Middle	5/5	5/5	0/0	5/5	Stars & Ducks	47/60	0/0	8/10	*79%*
Right	5/5	0/0	5/5	5/5	Stars & Plants	0/0	24/30	9/10	*83%*
Total	*100%*	*100%*	*100%*	*100%*	***Total by Object***	*85%*	*88%*	*85%*	

Table 6–2 Identification Results. (Left) Training results show that all fingerprints successfully identified all test objects in 2000 identification steps. (Right) Test results show that over 75% of the test object were successfully identified by the fingerprints in 2000 identification steps.

Entries in Table 6–2 summarize the number of objects tested across all seeds in the form of the ratio of successful identifications over the total number of tested objects. For learning environments, the denominator of 5 denotes the 5 tested seeds. For test environments, the denominator includes the product of the number of objects in the test environments and the number of stored fingerprints of that object type.[9] Any entry with 0/0 indicates that the object was not present in the identification environment. Blank spaces were omitted from the evaluation of the test environment due to the number of objects present. The final column and final row contain total percentages by test environment and object respectively. These results show that all fingerprints successfully identified the learned objects in their training environments and over 75% of all fingerprints successfully identified the learned objects in the test environments. By allowing the identification process to actively collect data where the fingerprint beliefs are uncertain, this percentage could be increased.

[9] *e.g.*, only one training environment has a star, so the denominator of the last entry in the first row is 5. The stars & plants environment contains 2 stars, so the denominator for star is 10.

6.4 Ablation Experiments

In this section, we perform ablation studies to assess the robustness of our active learning algorithm to changes in the sensor data content and to evaluate the influence of the learning hyperparameters. To modulate the sensor data content, we test learning at three different fixed camera heights z (see Fig. 6–13). We maintain the same test workspace for all tests, so the higher the camera is, the smaller the objects appear in the images. We also test adding z an additional conditional variable. We test the following hyperparameter combinations:

(1) fixed ramp β and $\gamma = 0$,

(2) adaptive β and $\gamma = 0$,

(3) fixed ramp β and adaptive γ, and

(4) adaptive β and adaptive γ,

where β is the latent space regularization hyperparameter from (6.34) and γ is the conditional reconstruction hyperparameter from (6.34).

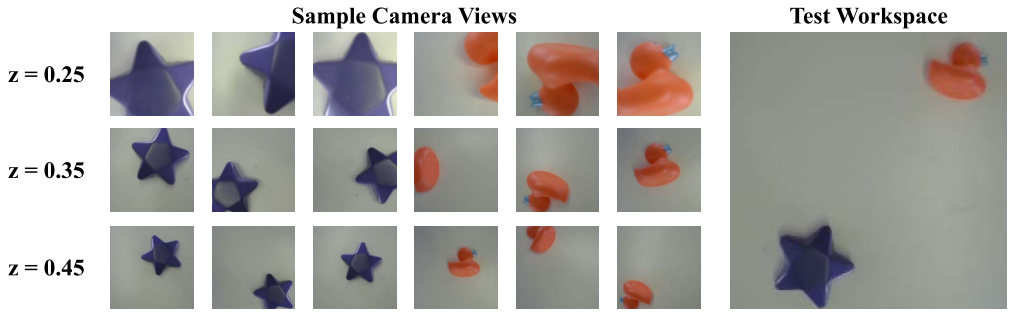

Figure 6–13 RGB Ablation Sample Sensor Data. Each row shows data collected from a different fixed z height. For tests with conditional z, the workspace spans $0.2 < z < 0.5$.

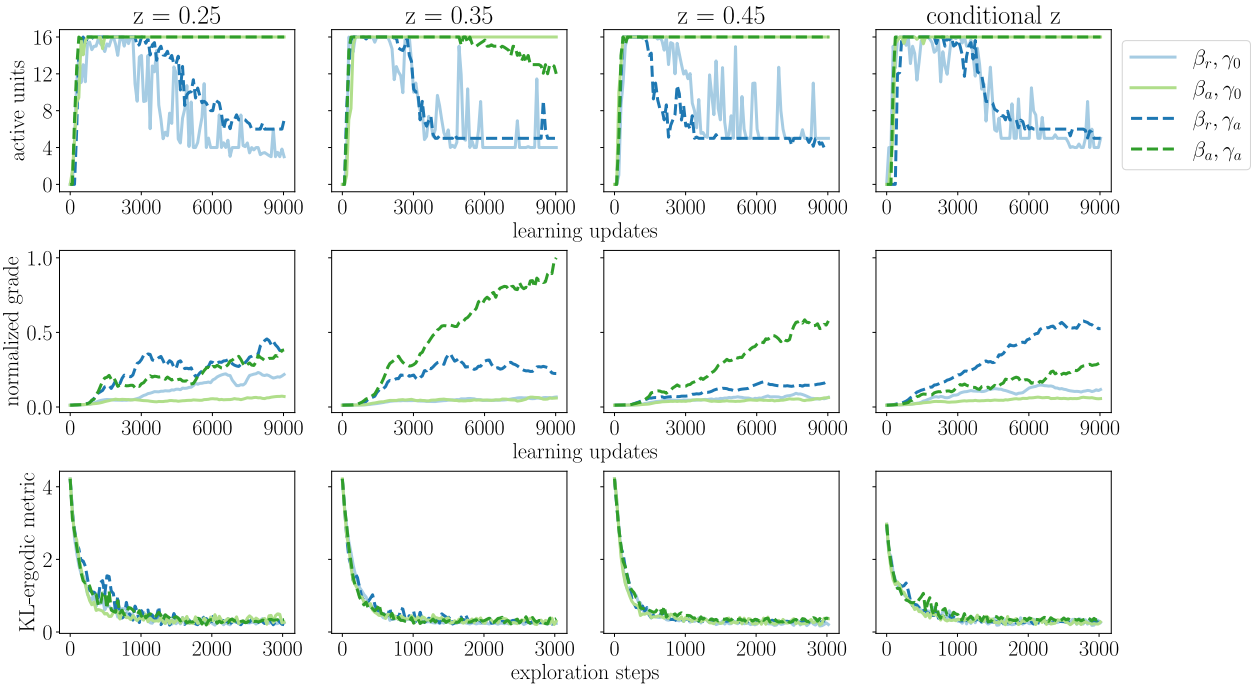

Figure 6–14 Ablation Learning Metrics. The first three columns summarize the learning metrics for tests with fixed z heights, and the last column includes z as a conditional variable. The top row shows the number of active units for each mini-batch update. Zero active units would indicate latent space collapse, and 16 active units indicates full latent space saturation. These plots show that the adaptive β tests maintained more active units throughout training than tests with a fixed ramp. The middle row shows the normalized grade of the conditional entropy map across learning updates. As the model learns to differentiate objects in the environment, the grade increases. These plots show that the adaptive γ tests resulted in a steeper grade than tests with $\gamma = 0$. The bottom row shows the ergodic metric for each exploration step—representing how well the collected data matches the conditional entropy distribution. These plots show that all methods were able to collect data that quickly converged to the conditional entropy distribution.

6.4.1 Learning

First we evaluate the impact of ablations during learning across three learning metrics—active units, normalized grade, and the KL-ergodic metric. Since β controls the influence of the latent space regularization, we expect β to have a greater impact on active units than γ. Since γ controls the influence of the conditional reconstruction loss, we expect γ to have a greater influence on the conditional entropy grade than β. We expect all active learning methods to be able to match their data collection to the conditional entropy distribution,

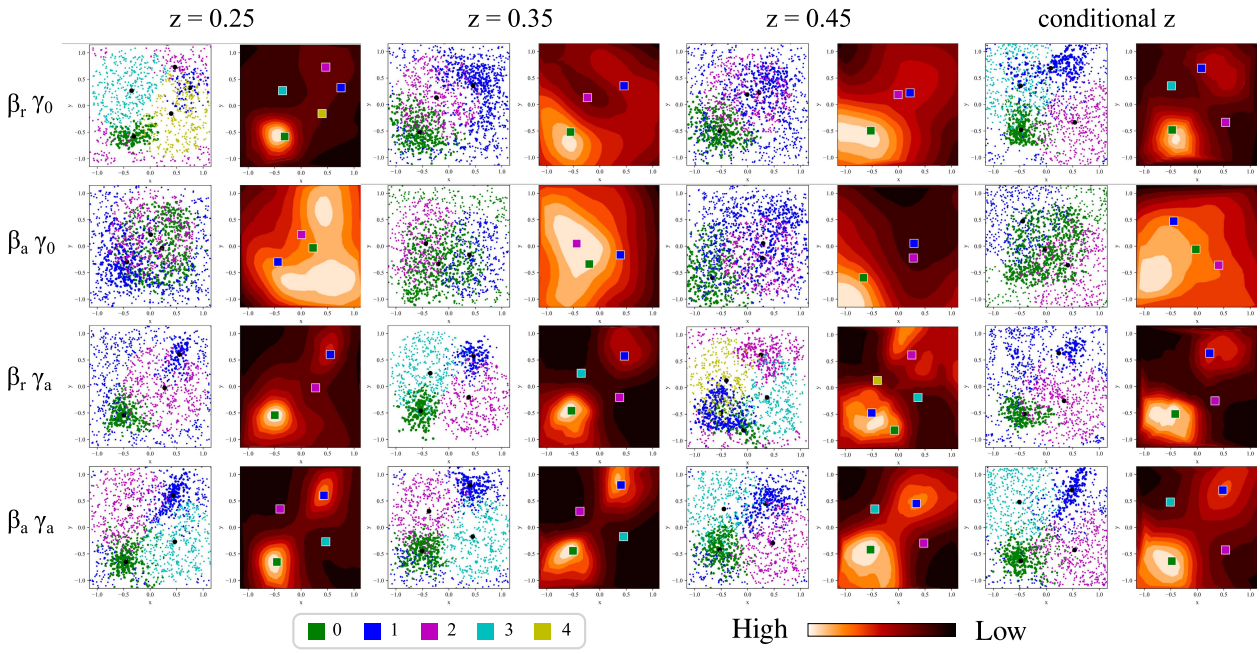

Figure 6–15 Ablation Clustering. The first three columns illustrate clustering for tests with fixed z heights, and the last column includes z as a conditional variable. Each row shows a different hyperparameter ablation condition. The heatmaps show the final conditional entropy distributions for each ablation test (with a fixed yaw for visualization purposes). The colored squares correspond to the center of objects extracted from the conditional entropy. The colors match the samples on the paired plots. Samples were generated from the conditional entropy distribution as well as the inverted conditional entropy distribution. These plots show that with $\gamma = 0$, the number and distribution of clustered objects varies substantially across different z values. With adaptive γ, the clusters are much more consistent across heights, but the most separable distributions are generated with both adaptive β and adaptive γ.

regardless of the CVAE loss function. Fig. 6–14 shows the evolution of our ablation tests. Each column shows the learning metrics for different z heights. Each plot shows curves for each ablation condition. As expected, these results show that the β hyperparameter primarily impacts the number of active units. Across all heights, the ramped β tests converged to approximately half as many active units as the actively tuned β tests. Additionally, the normalized grade results show that γ impacts the conditional entropy grade more than the β method. Finally, the results show that all tests were able to collect data that quickly converged to the conditional entropy distributions.

6.4.2 Fingerprinting

During fingerprinting (as was described in Section 6.2.4), we determine object locations using clustering of the conditional entropy. In this section, we assess the impact of the hyperparameters on clustering. The learning metrics results show that γ has a greater influence on the conditional entropy grade than β. Maximizing the grade is important for two reasons related to clustering. First, the steeper the grade, the more strongly the controller is encouraged to collect data in the vicinity of the objects rather than throughout the reachable workspace. This allows the model to better satisfy the *i.i.d.* requirements of the loss function. Second, the more differentiated the objects are, the easier it is to identify objects during clustering. Fig. 6–15 shows the conditional entropy distributions for each ablation test as well as the clustered objects. As expected, the adaptive γ tests have more consistent clusters across tests than those with $\gamma = 0$. These results indicate that β also has an influence on the conditional entropy distributions and by extension the clusters. The adaptive β clusters are the most consistent across test conditions and result in more visibly distinguishable clusters.

6.4.3 Identification

During identification (as was described in Section 6.2.5), we build likelihood models using the newly collected images and the stored fingerprint locations. In this section, we qualitatively assess model reconstructions to assess whether or not our stored fingerprints are suitable for identification.

Figs. 6–16, 6–17, and 6–18, show results from testing the model with real and fake model input seeds. To test model input seeds, we generate a latent space with a particular image

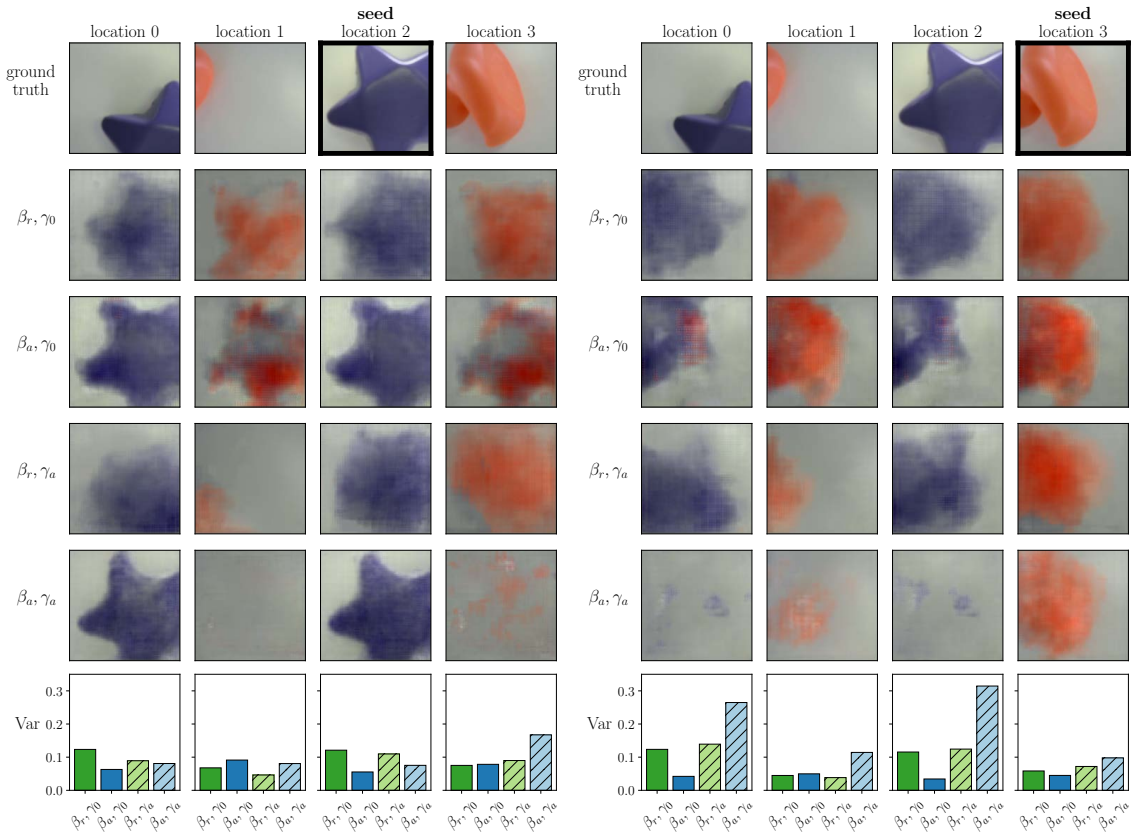

Figure 6–16 CVAE ablation results (seeds 2 and 3 $z = 0.25$). The top row shows the ground truth image for each subsequent row. The seed image and state is indicated with a thick black box. The next four rows show reconstructions for each loss hyperparameter ablation. The final row shows the predicted variance for each image. These results show that when $\gamma = 0$, the model tries to project the learned object onto the image of the seed object. When using adaptive γ, the model tries to reproduce the learned image, while the seed conditions how confident the model is in its prediction (*i.e.*, the variance).

and state. That latent space is then decoded at the conditional states specified by the ground truth locations of the images in the first row. The next four rows show reconstructions for each of our four ablation methods, and the final row shows the predicted variances for each method. In this section, we discuss results for $z = 0.25$. See Appendix C for ablation reconstruction results for other seeds and heights.

For Fig. 6–16, the model is seeded with the true image and location of seed 2 (left) and seed 3 (right). These results suggest that the latent space plays a role in the reconstruction,

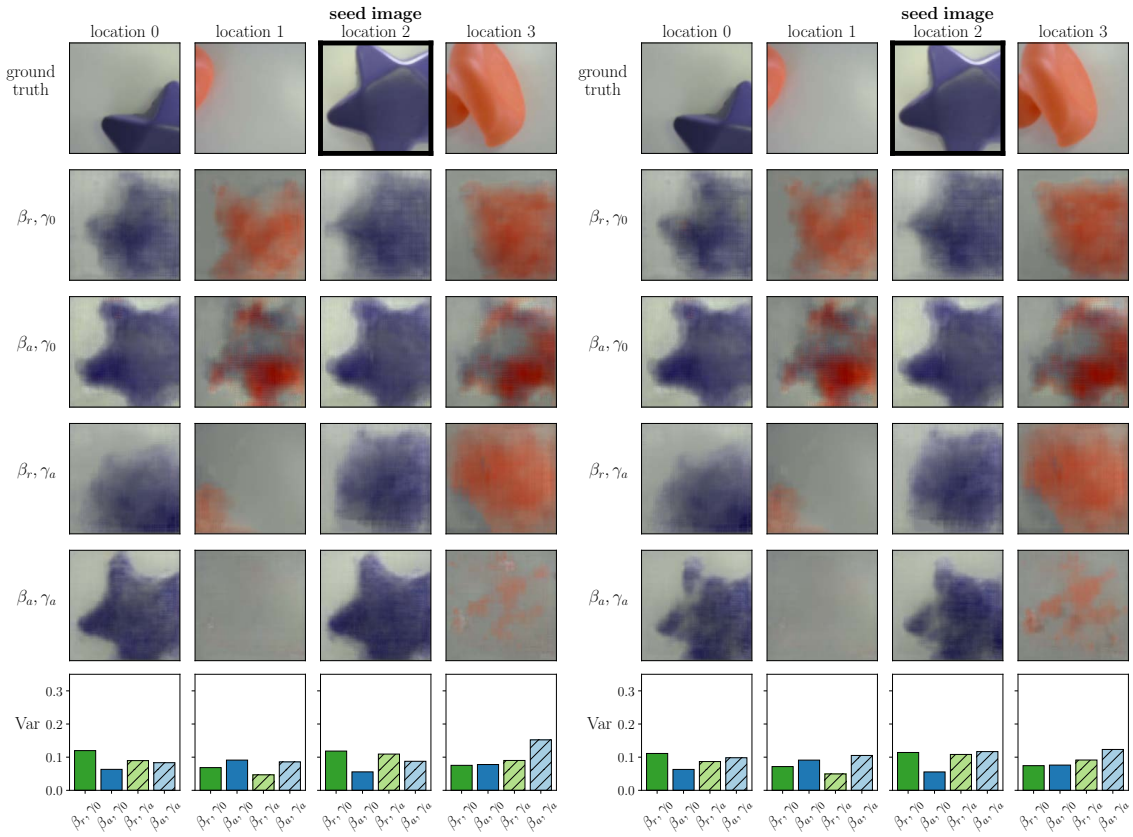

Figure 6–17 CVAE state ablation (seed 2, $z = 0.25$). The top row shows the ground truth image for each subsequent row. The seed is indicated with a thick black box. On the left side, the seed state is replaced with all 0's. On the right side, the seed state is replaced with all 1's. The final row shows the predicted variance for each image. These results are nearly identical to the predictions from Fig. 6–16, which suggests that the encoder ignores the input state after learning.

and the decoder did not simply memorize the images at the conditional states. If the decoder had memorized the states, the reconstructed images would look exactly like the ground-truth images, regardless of the model input seeds. Instead, the results show that for all methods, the test locations closer to the seed location have better reconstructions (*i.e.*, closer to ground truth) with lower variance than objects further away from the seed. Additionally, these results show that the test with adaptive β and adaptive γ produces a larger range of variances across test locations, which is consistent with the grade results from the learning section as well as the clustering results.

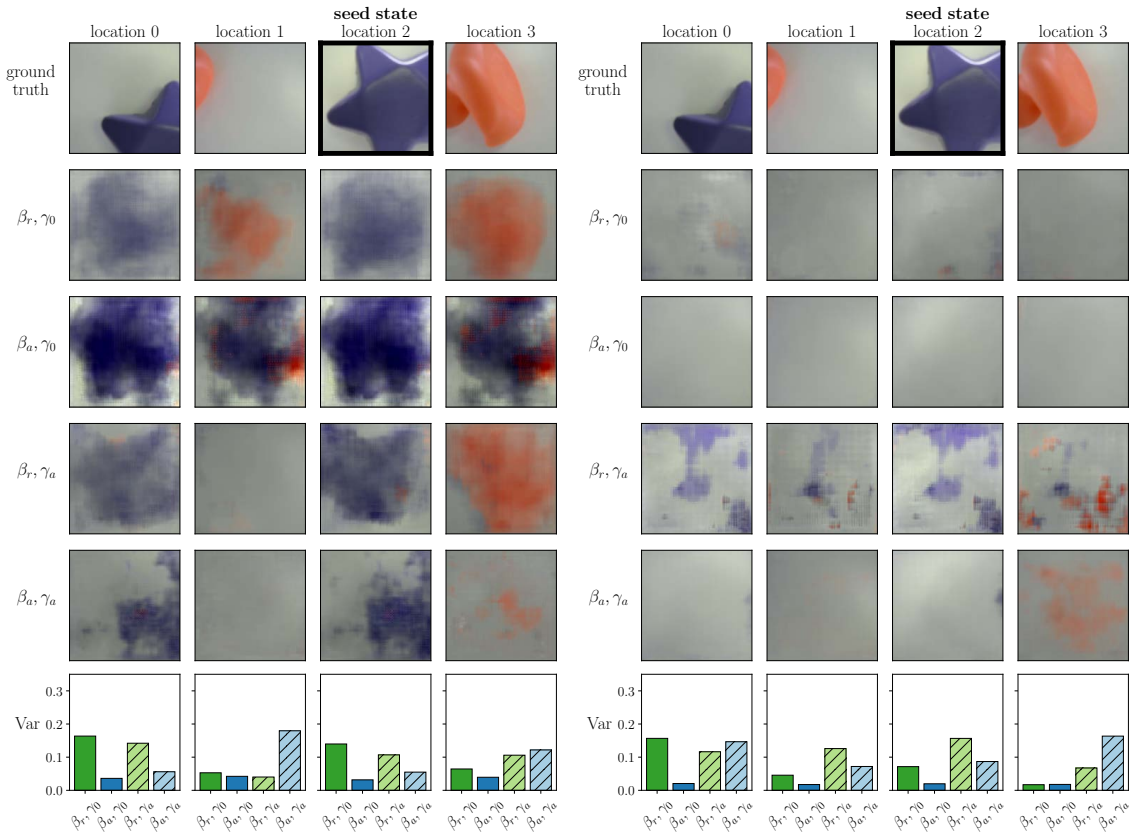

Figure 6–18 CVAE image ablation (seed 2, $z = 0.25$). The top row shows the ground truth image for each subsequent row. The seed is indicated with a thick black box. On the left side, the image is replaced with all 0's (black image). On the right side, the image is replaced with all 1's (white image). The final row shows the predicted variance for each image. These results show that the encoder is highly conditioned on the input image. Similar to Fig. 6–16, when $\gamma = 0$, the model tries to project the learned object onto the image of the seed object. The adaptive γ model tries to reproduce the learned image, but the model has greater variance (more uncertainty) when given an ablation seed image, which is much different from the learned image.

While Fig. 6–16 confirmed the latent space plays a role in image reconstruction, Figs. 6–17 and 6–18 aim to assess the impact of the state and the image data individually. Fig. 6–17 tests real seed images with fake seed states. The left side uses a seed state of all 0's, and the right side uses a seed state of all 1's.[10] By inspection, the reconstructions in this figure are nearly identical to the reconstructions in Fig. 6–16. This suggests that after learning, the encoded latent space is primarily a result of the input image.

[10]Workspace states range from [-1,1], where the duck is closest to corner [1,1].

Fig. 6–18 tests real seed locations with fake seed images. The left side uses a seed image of all 0's (black), and the right side uses a seed image of all 1's (white). The reconstructions in this figure are much different from those in the prior figures. This suggests our identification method of testing new images with stored states should produce different results when the images match the trained locations versus when they do not.

6.5 Experiments with Additional Sensors

A benefit of the proposed learning framework is that it does not matter what sensors are provided as sensory data. Since there is no explicit model of the sensor, it should not matter if the sensor data is generated by a camera, a microphone, a soft robot, or a combination of sensors. In this section, we demonstrate the learning method with two additional sensors—a GelSight touch sensor and an ultrasound imager. These modalities both output images, so we can use our same camera pipeline, but data are quite different from the RGB camera data. RGB cameras generate smooth data transitions between observable states throughout the workspace. Meanwhile, touch sensors can experience data discontinuities between no-contact and contact conditions. Ultrasound sensors have a similar transition between in-air and in-phantom conditions with the added complexity of non-linear sensor dynamics.

6.5.1 GelSight Touch Sensor

For touch experiments, sensor data was collected using two sensors—a GelSight Mini tactile sensor and a force/torque sensor. The GelSight Mini sensor (GelSight Mini, GelSight, Waltham, MA, USA) has an internally mounted camera, RGB LEDs, and a fingertip gel cartridge (see Fig. 6–19). The gel cartridge has an interaction surface area of roughly 20mm

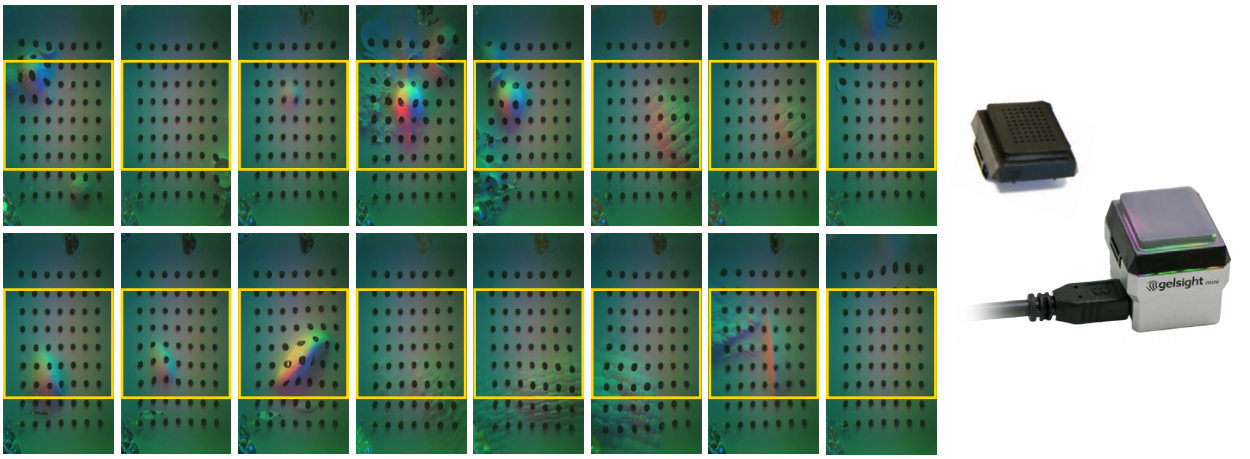

Figure 6–19 Sample data collected with a GelSight Mini touch sensor (right). Yellow boxes indicated cropped portions of sensor data that are provided to the CVAE.

× 25mm. For our experiments, we use a gel cartridge, which has tracker dots embedded in the gel surface. As the sensor interacts with objects, the gel is perturbed, and the camera captures this deformation. To achieve maximally informative measurements, the sensor surface must be tangent to the object's surface.

Compared to the RGB webcam, the GelSight is a relatively immature technology that is susceptible to damage during operation. The edges of the gel cartridge are damaged most often, so for our experiments we resize the field of view of the sensor to remove the edges of the image from the measurement (see Fig. 6–19). Another common failure mode is for the cable to be dislodged from the sensor during testing. To reduce the likelihood of damaging the sensor, we restrict the conditional states to minimize the chances that the robot will damage the cable. We also add a physical barrier around the cable, which increases the geometry of the robot's end effector (see Fig. 6–20).

For our tactile experiments, we tested the three environments shown in Fig. 6–20. The tactile sensor needs to be able to conform to the surface of the object, so the robot has access to x, y, z position as well as end effector pitch in the world frame. We also included z velocity

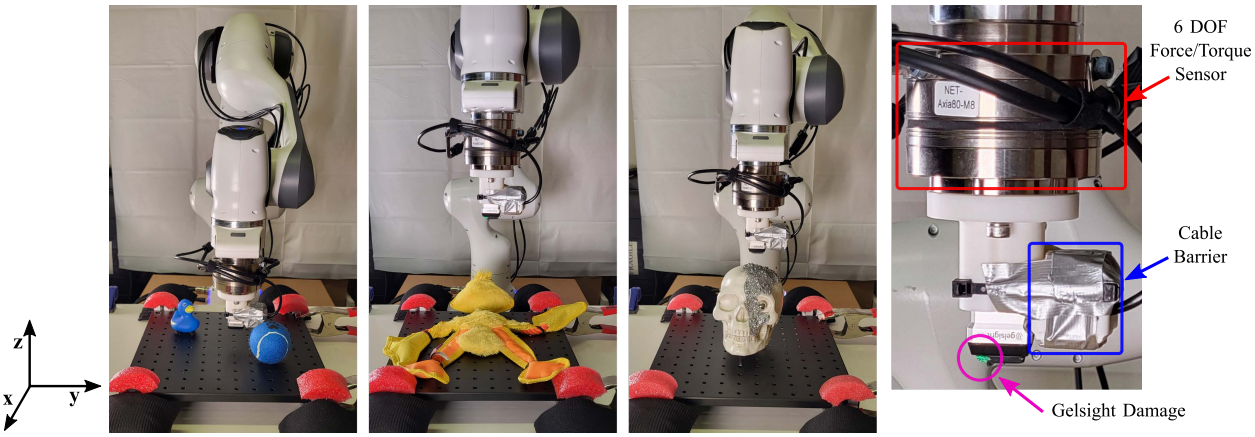

Figure 6–20 GelSight Test Environments. From left-to-right, we call these environments "Ball + Duck", "Dog Toy", and "Skull". The "Ball + Duck" is most similar to our RGB camera experiments because there are discrete objects in the test workspaces, which can be separated in the x-y plane. The "Dog Toy" environment has features spread throughout the environment, but different regions of the toy have different soft textures (the head and tail are fully soft, but the other portions of the toy have a rope embedded below the surface). The "Skull" environment has a single object with various hard features. The skull is much larger than the other test objects, and as a result, this environment contains the largest volume of unreachable workspace states. The far right image provides a zoomed-in view of the robot's end effector with the force/torque sensor, cable barrier, and GelSight. The light visible in the circled region demonstrates what gel cartridge damage looks like externally.

as a conditional variable as a proxy for controlling contact force, so the full conditional state includes 5 dimensions. Because the tactile sensor has a relatively small contact area, end effector can bump into objects or the table without tactile sensor contact. Therefore, we also include end effector force as a sensor measurement. The robot receives one force value, which is the norm of the x, y, z forces. The CVAE outputs a single uncertainty value for both sensor measurements.

Fig. 6–21 summarizes the learning metrics for the three environments shown in Fig. 6–20. As was discussed in the experiment introduction, a major difference between the touch experiments and the camera experiments is contact. With all three touch environments, there are regions of the workspace that the robot cannot access due to the physical volume of the objects. Although all experiments learned active latent spaces, the purple curves

show that the original learning method from Algorithm 6–1 does not appear to result in good performance in terms of the KL-ergodic metric. Since the robot is unable to visit some regions of the workspace, it is unable to collect data from those regions.

To incorporate the unreachable workspace into the controller, we add a "avoid distribution" into the controller formulation. The avoid distribution is generated using the same method that we used to generate the target distribution from (6.31), but the avoid distribution only includes visited states that exceed a force threshold.[11] By subtracting the avoid distribution from the conditional entropy target distribution, the algorithm collects more data from the surface of the objects and is able to achieve a better KL-ergodic metric.[12]

Fig. 6–21 also includes a visualization of the conditional entropy distribution for each test environment as well as an alternate view of the workspace. By inspection, we can see that the "Ball + Duck" model learned to differentiate the locations of the objects from the empty space in the test workspace. The "Dog Toy" environment has information throughout the x-y plain, but qualitatively, the high regions of the conditional entropy map onto the head and feet regions of the toy and include higher information content where the toy is present than in blank regions. Finally, the high regions of "Skull" conditional entropy align with the object in the plane but examining slices would likely be necessary to extract features across the surface of the object. Future work should examine methods of clustering for touch that are conditioned on the observed features of the learned objects rather than the states of the test environment as a whole.

[11] We used a state coverage Gaussian 10x smaller than the Σ used for trajectory coverage.

[12] Since we are using a sample-based method of calculating the ergodic metric, we note that an alternate method of incorporating the avoid distribution into the control problem could be to use avoid distribution for rejection sampling of reachable workspace samples.

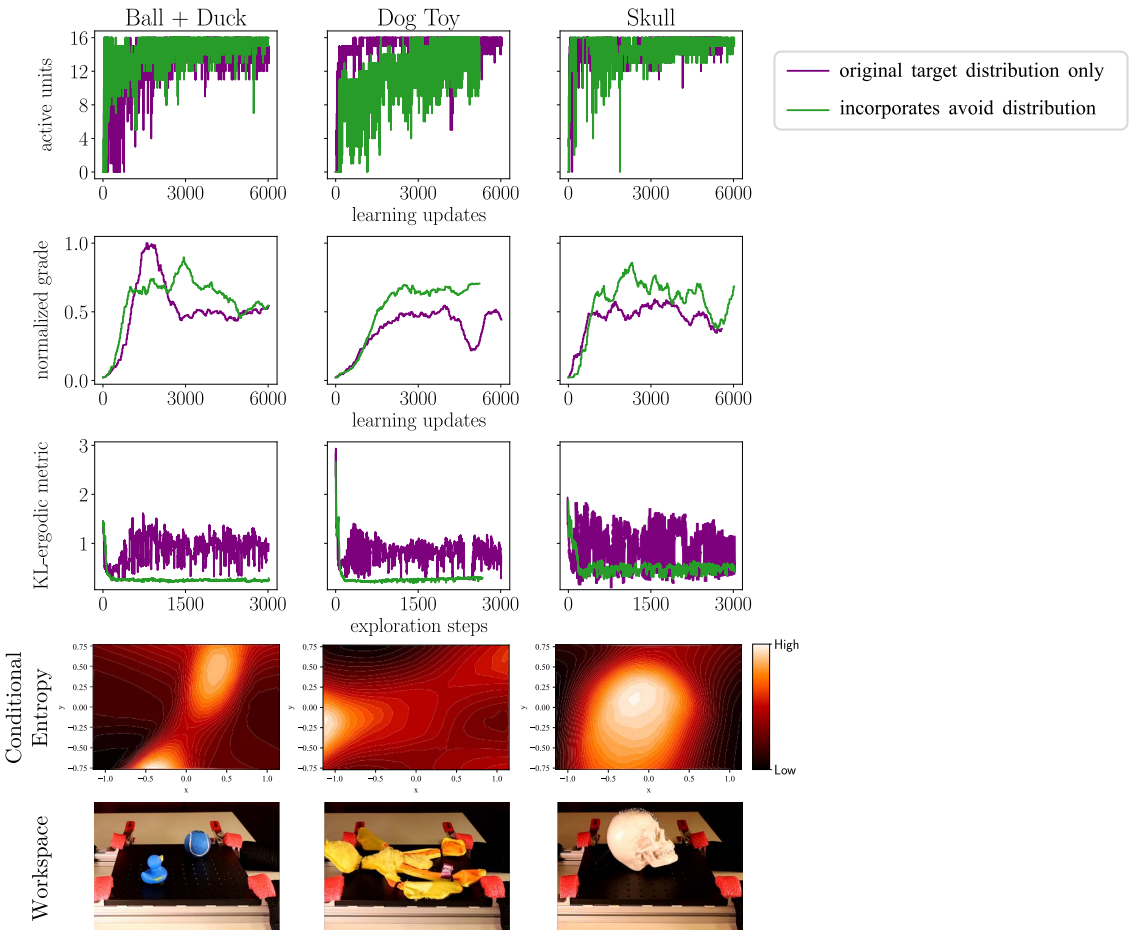

Figure 6–21 GelSight Learning. For our GelSight experiments, we tested two exploration methods per environment. The "original target distribution only" uses the baseline ergodic control method, which has no method for avoiding unreachable regions of the workspace. The "incorporates avoid distribution" also builds an *avoid* distribution during learning to track unreachable regions of the workspace (determined by measured force), and incorporates the unreachable regions into the target distribution. The top row shows the number of active units for each learning environment. Zero active units would indicate latent space collapse, and 16 active units indicates full latent space saturation. The middle row shows the normalized grade of the conditional entropy map across learning updates. As the model learns to differentiate objects in the environment, the grade increases. The third row shows the ergodic metric for each exploration step—representing how well the collected data matches the conditional entropy distribution. The fourth row shows the conditional entropy distribution, and the final row shows the workspace aligned with the conditional entropy distribution. These plots show that without incorporating knowledge of unreachable regions, the ergodic agent has difficulty minimizing the ergodic metric. Nonetheless, all methods were able to build conditional entropy distributions matching the distribution of features in the environment while maintaining a large number of active units.

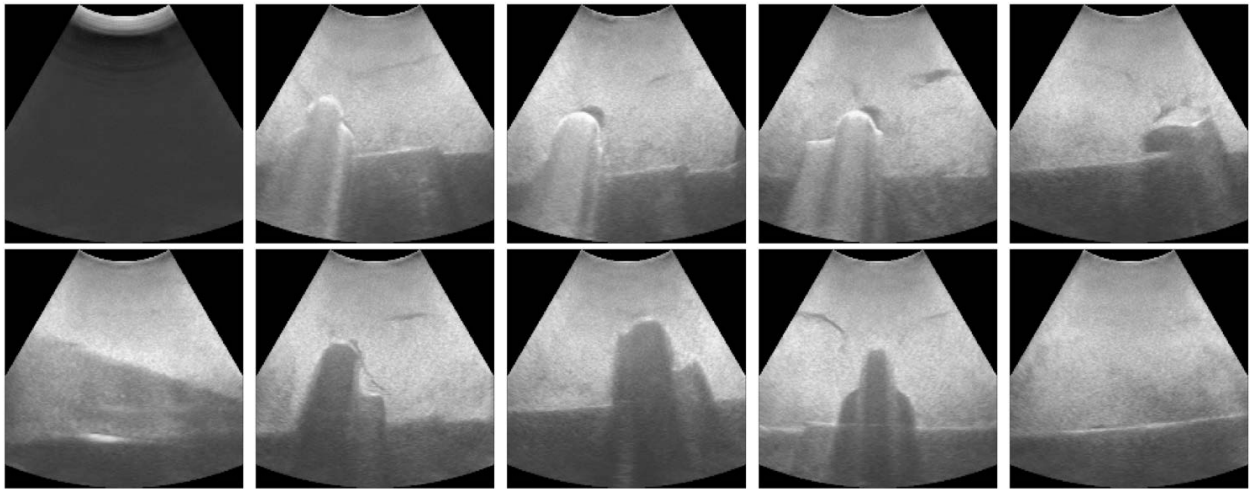

Figure 6–22 Sample data collected with the ultrasound system. The top left image is an example of the ultrasound data when not in contact with the jello. The light color duck is the metal duck.

6.5.2 Ultrasound Imager

For ultrasound experiments, sensor data was collected with a duplex ultrasound imaging device (ArtUs EXT-1H, Telemed Medical Systems, Vilnius, Lithuania). The ultrasound system was equipped with a convex array transducer (C5-2H60-A5, Telemed Medical Systems), which operates at 2.0-5.0 Hz with a 65 deg/mm field of view. We operated the transducer at a B-mode imaging frequency of 5 MHz using the ArtUs RF Data Control for Python package (version 1.0.1, Telemed Medical Systems). Ultrasound requires objects to be embedded in a dense medium for the sound waves to travel through. Synthetic ultrasound test volumes are often called phantoms. For our experiments, we use the jello phantom generation method outlined in [**216**]. The jello phantom is shown in the middle of Fig. 6–23. Our phantom had 3 embedded objects—a rubber duck filled with molasses, a rubber duck filled with screws and washers, and a solid metal duck. Sample ultrasound images are shown in Fig. 6–22.

For our ultrasound experiments, the robot had access to conditional states x, y, z and yaw. The z workspace included regions in air as well as regions within the jello phantom. The

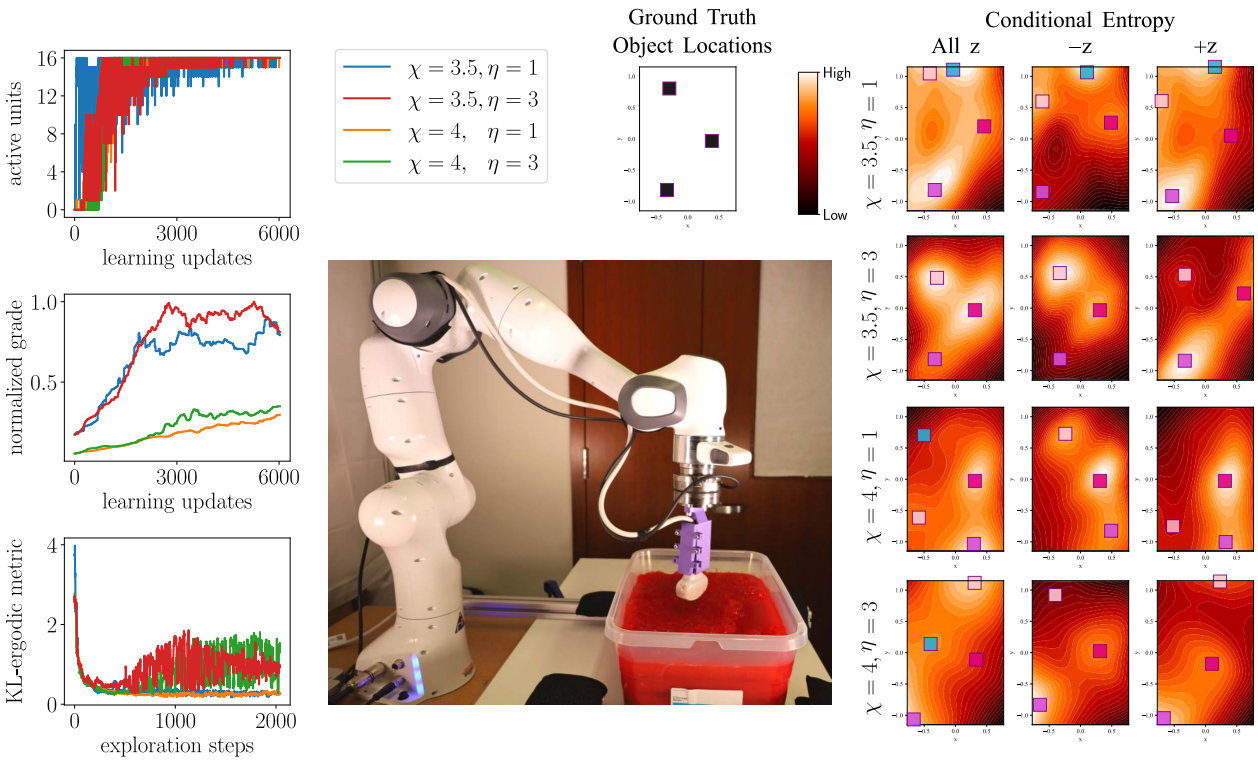

Figure 6–23 Ultrasound Learning. For our ultrasound experiments, we tested four different hyperparameter configurations. The left side of the figure shows learning metrics for each configuration. The top row shows the number of active units for each learning environment. Zero active units would indicate latent space collapse, and 16 active units indicates full latent space saturation. The middle row shows the normalized grade of the conditional entropy map across learning updates. As the model learns to differentiate objects in the environment, the grade increases. The bottom row shows the ergodic metric for each exploration step—representing how well the collected data matches the conditional entropy distribution. The middle of the figure shows the jello phantom workspace (with 3 embedded ducks) and the ultrasound attached to the robot's end effector. These learning metrics show that although all methods were able to maintain a high number of active units, those with a lower χ value resulted in a steeper normalized grade. Methods with an additional The right side of the figure shows conditional entropy distributions across different z values. As is shown in Fig. 6–22, the ultrasound data is very different when in contact with jello than when in contact with air. To demonstrate the impact on the conditional entropy distribution, we display three different conditional entropy distributions. The first column is averaged over all values of z, the second column only includes values of z that are submerged in the jello, and the third column includes values of z in air. These The fourth row shows the conditional entropy distribution, and the final row shows the workspace aligned with the conditional entropy distribution. These plots show that without incorporating knowledge of unreachable regions, the ergodic agent has difficulty minimizing the ergodic metric. Nonetheless, all methods were able to build conditional entropy distributions matching the distribution of features in the environment while maintaining a large number of active units.

top left image of Fig. 6–22 shows an example of ultrasound data in air, while the remaining images are in contact with the jello. Because the robot can interact with the jello, the sensor data can change over time. When the robot passes quickly through the jello, it can leave behind air bubbles that add additional noise to the data. If jello sticks to the ultrasound probe when it emerges from the jello, the field of view in air may include speckled regions rather than the mostly-black image shown in the top left of Fig. 6–22.

Fig. 6–23 summarizes the learning metrics for the ultrasound tests. Since ultrasound has different physics than an RGB camera, we tested four different hyperparameter conditions. We tested two different values of χ, which controls the minimum grade from (6.35). We also tested two different values of η, which scales the workspace coverage from (6.33). Although all experiments learned active latent spaces, the curves in the normalized grade plot show that the tests with lower χ resulted in a steeper grade, which encourages the robot to collect data around the detected objects earlier during training. The curves in the KL-ergodic metric plot show that the tests with $\eta = 3$ had more noise than those with $\eta = 1$. In this case, the noise was a result large differences in the conditional entropy across different locations in workspace. Since the sensor data has a different background entropy in-air vs in-jello, we plot three different conditional entropy distributions per test condition. The right side of Fig. 6–23 shows three conditional entropy maps–the first column averages across all z values, the middle column averages across negative z values (*i.e.*, in jello), and the last column averages across positive z values (*i.e.*, mostly in air). Collectively, the high regions of the conditional entropy match the object locations in the environment. Future work should explore methods of automatically separating the in-air and in-jello environments prior to clustering.

6.6 Discussion and Conclusion

In this chapter, we presented a closed-loop embodied learning method that does not require pre-existing datasets, offline compute, or communication to the cloud to learn latent perception models. Instead, by exploiting control and the subsequent ability to shape the data it acquires, the robot regulates the learning process through data collection, reasoning about needed data based on the evolving state of the CVAE at every learning iterate. In contrast to next-best-view, our approach creates a sequence of high quality views at every time step, with subsequent guarantees on diversity of data (since no two views can be the same) and domain coverage. Moreover, our approach does not suffer from the catastrophic collapse of the latent space one gets when using randomized search across the domain. The method creates generative models of the domain that can be used to separate features into distinct objects. Furthermore, the generative models can then be used to search for the objects. Lastly, as might be expected from a technique that uses continuous feedback from the learning model as it evolves, performance varies little from experiment to experiment—every data collection trial leads to generative models with information-rich latent spaces for search and identification.

The method introduced in this chapter demonstrated the ability to learn in single-shot embodiments across different conditional variables and sensor modalities, but there are still several limitations. First, we assume the test environment is fixed during learning. Future work could investigate learning in dynamic environments. Another limitation is our use of end effector control during our experiments. To fully explore in all six degrees of freedom, the robot will require knowledge of joint limits, so joint-control will likely be required.

Additionally, our fingerprinting method naively collects data in the vicinity of object locations (as identified by clustering of the conditional entropy). Future work could investigate methods of active fingerprinting to collect maximally informative fingerprints. Similarly, the identification method generates workspace beliefs over a fixed grid. Future work could investigate alternate belief specifications, with better scaling properties. Finally, the identification method currently only uses the latent space parameters for the measurement model, but future work could also incorporate the uncertainty of the reconstructed image as part of the measurement model.

Chapter 7

Conclusions

The presented work focuses on overcoming the limitations of current Reinforcement Learning (RL) methods and building perception models to better equip robots for physical operations, but the lessons are broadly applicable to Machine Learning (ML). We need to move away from the pristine world of simulated environments and clean data. Until we do so, we are holding back our ML methods from reaching their full potential. The way we explore and collect data has deep implications for the learned model, and this work will expand our understanding of the untapped potential of actively learned models, subject to the vagaries of the real world. By taking a step away from the rigid train-test paradigm and a step toward a continual, single-shot active learning framework, we will be able to build models better-suited to the physical world—not just for robot learning—but for ML at large.

7.1 Computation

Access to computation can influence the types of algorithms we develop. For the purposes of this discussion, I'll divide compute into three categories. *Big* compute involves offline, cloud computing with massive capabilities, but also requires constant, high-speed network connections. *Medium* compute involves an on-site desktop or server capable of fast learning

and rapid communication to other local systems. *Small* compute encompasses laptops and other system, which could be placed on-board robotic systems.

Big, offline cloud computing is not well suited to embodied learning, particularly when sensor data is involved. There is a time cost associated with uploading sensor data to a server and a connectivity cost imposed by big compute. Ultimately, we want our robots to be able to operate independently and be responsive to evolving conditions in real time, so being able to close the loop between our computation and our robots is essential. Similarly, *small* compute is not well suited to embodied learning prototyping. Insufficient compute during prototyping can result in designing algorithms on short horizons, small models, or other concessions, which can lead the developer to falsely conclude that certain algorithms are not suitable for their task or modifying certain parameters isn't relevant because they are unable to sufficiently test the limits of their methods.

In this work, I benefited from access to *medium* compute during the prototyping phase, which enabled me to develop a distributed method of active robot learning, which coupled the state of the learning system to the robot's data collection process. I was able to not only learn models in real-time, but also develop dashboards to monitor the evolution of the process during learning. These tools enabled me to gain intuition about the interplay between learning systems, sensors, and various parameters, and ultimately led me to develop many of the theoretical contributions of this work. Although many of these algorithms can now run on *small* compute, access to *medium* allowed flexibility throughout the development process to investigate various aspects of the algorithms.

7.2 Future Directions

While developing the work presented in this thesis, I observed interesting avenues for future investigation that I didn't have time to pursue. In this section, I outline a few of these ideas for possible extensions and future directions.

In Chapter 4, we discussed how single-shot deployments can enable real-world learning. During our single-shot experiment development, we observed that by tracking the entropy of the gradients of the network during training, we could detect when an agent learned a particular behavior. For example, during swimmer experiments we could detect when the algorithm learns to swim as well as instances where catastrophic forgetting occurs (where the algorithm forgets how to swim and starts learning from scratch). Future work could investigate methods using the network learning dynamics to improve learning. Perhaps events such as catastrophic forgetting could be detected in advance. Or perhaps the gradients could be used to anneal the temperature-like parameter from Maximum Diffusion Reinforcement Learning (MaxDiff RL).

In Chapter 6, we discussed how the ergodic control can enable robots to collect *i.i.d.* data and extend PAC-Bayes bounds for VAEs to CVAEs. Future work could develop methods of evaluating whether or not the bounds are actually satisfied online for our specific context. Furthermore, the bounds from [**84**] formally assume the decoder has a fixed variance, but our method learns the decoder variance as well. Future work could investigate what modifications are formally required to incorporate the learned variance into the PAC-Bayes framework.

The perception learning examples presented in Chapter 6 operate in relatively simple proof-of-concept environments. Throughout this work, I was motivated by the idea of using our method to identify a particular duck in a pile of many rubber ducks. Accomplishing this

task would likely require joint control (and possibly safety controllers) to allow the robot to efficiently learn in high dimensions. Furthermore, the robot may need to be able to interact with the objects to fully reduce its uncertainty.

An additional benefit of generative perception models is that they can enable us to teach robots behaviors rather than tasks. For example, how would you teach a robot the benefit of using a tool like a flashlight? This lesson is more difficult to formulate as a task than teaching a robot to turn on a light (but also more interesting). Shining a flashlight across the surface of an object reveals different features of an object and this is true for most objects, not just a particular object. With our method, lighting conditions (like control of a flashlight) can simply become another conditional variable in the CVAE framework. Not only might this help the robot recognize the object in the future, but it also teaches the robot how to use a tool to manipulate the learning environment. Light could also play a role in helping discriminate the difference between identical objects with different surface finishes. For example the way light reflects off the surface of a rubber duck would be very different from the surface of an aluminum duck. Future work could extend our method to investigate learning robot behaviors. Ultimately, these examples are only a few of many possible extensions of this work which could expand the capabilities of our perception learning methods to accommodate the messiness of real-world scenarios.

Bibliography

[1] A. R. Ansari and T. D. Murphey, "Sequential action control: Closed-form optimal control for nonlinear and nonsmooth systems," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1196–1214, 2016.

[2] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[3] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *arXiv*, 2018.

[4] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," *arXiv*, 2018.

[5] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[6] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[7] W. H. Montgomery and S. Levine, "Guided policy search via approximate mirror descent," in *Advances in Neural Information Processing Systems*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016.

[8] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[9] I. Abraham, A. Broad, A. Pinosky, B. Argall, and T. D. Murphey, "Hybrid control for learning motor skills," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2021.

[10] A. Pinosky, I. Abraham, A. Broad, B. Argall, and T. D. Murphey, "Hybrid control for combining model-based and model-free reinforcement learning," *International Journal of Robotics Research*, vol. 42, no. 6, pp. 337–355, 2023.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, 2018.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv*, 2016.

[13] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[14] T. A. Berrueta, A. Pinosky, and T. D. Murphey, "Maximum diffusion reinforcement learning," *Nature Machine Intelligence*, vol. 6, no. 5, pp. 504–514, 2024.

[15] A. Prabhakar and T. Murphey, "Mechanical intelligence for learning embodied sensor-object relationships," *Nature Communications*, vol. 13, 2022.

[16] A. Pinosky and T. Murphey, "Embodied active learning of generative sensor-object models," in *International Symposium of Robotics Research (ISRR)*, 2024.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *International Journal of Robotics Research*, vol. 40, no. 4, pp. 698–721, 2021.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, 2017.

[20] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[21] I. Abraham, A. Handa, N. Ratliff, K. Lowrey, T. D. Murphey, and D. Fox, "Model-based generalization under parameter uncertainty using path integral control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2864–2871, 2020.

[22] A. Havens, Y. Ouyang, P. Nagarajan, and Y. Fujita, "Learning latent state spaces for planning through reward prediction," *Advances in Neural Information Processing Systems*, 2019.

[23] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman, "Dynamics-aware unsupervised discovery of skills," *International Conference on Learning Representations (ICLR)*, 2020.

[24] I. Abraham, G. de la Torre, and T. Murphey, "Model-based control using Koopman operators," in *Robotics: Science and Systems (RSS)*, 2017.

[25] I. Abraham and T. D. Murphey, "Active learning of dynamics for data-driven control using Koopman operators," *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1071–1083, 2019.

[26] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning (ICML)*, 2011, pp. 465–472.

[27] A. A. Taiga, W. Fedus, M. C. Machado, A. Courville, and M. G. Bellemare, "On bonus based exploration methods in the arcade learning environment," in *International Conference on Learning Representations (ICLR)*, 2020.

[28] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2778–2787.

[29] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[30] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3381–3388.

[31] S. Bansal, R. Calandra, K. Chua, S. Levine, and C. Tomlin, "MBMF: Model-based priors for model-free reinforcement learning," *arXiv*, 2017.

[32] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014.

[33] N. Lambert, B. Amos, O. Yadan, and R. Calandra, "Objective mismatch in model-based reinforcement learning," *Annual Conference on Learning for Dynamics and Control*, 2020.

[34] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *International Conference on Informatics in Control, Automation and Robotics*, 2004, pp. 222–229.

[35] G. Tang, W. Sun, and K. Hauser, "Learning trajectories for real- time optimal control of quadrotors," in *IEEE International Conference on Intelligent Robots and Systems*, 2018, pp. 3620–3625.

[36] K. Lu, A. Grover, P. Abbeel, and I. Mordatch, "Reset-free lifelong learning with skill-space planning," in *International Conference on Learning Representations (ICLR)*, 2021.

[37] A. Chen, A. Sharma, S. Levine, and C. Finn, "You only live once: Single-life reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[38] J. Oh, S. Singh, H. Lee, and P. Kohli, "Zero-shot task generalization with multi-task deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2661–2670.

[39] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of zero-shot generalisation in deep reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 76, pp. 201–264, 2023.

[40] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[41] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning (CoRL)*, 2022, pp. 991–1002.

[42] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," in *Robotics: Science and Systems (RSS)*, 2021.

[43] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2950–2957, 2020.

[44] H. R. Walke, J. H. Yang, A. Yu, A. Kumar, J. Orbik, A. Singh, and S. Levine, "Don't start from scratch: Leveraging prior data to automate robotic reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2022.

[45] L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine, "Legged robots that keep on learning: Fine-tuning locomotion policies in the real world," in *International Conference on Learning Representations (ICLR)*, 2022, pp. 1593–1599.

[46] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2020, pp. 1025–1037.

[47] A. Sharma, A. M. Ahmed, R. Ahmad, and C. Finn, "Self-improving robots: End-to-end autonomous visuomotor reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2023.

[48] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1988.

[49] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5628–5635.

[50] P. Florence, L. Manuelli, and R. Tedrake, "Self-supervised correspondence in visuomotor policy learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 492–499, 2019.

[51] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 661–668.

[52] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011, pp. 627–635.

[53] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2000.

[54] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2004.

[55] B. Mildenhall, P. Srinivasan, M. Tancik, J. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *European Conference on Computer Vision*, 2020.

[56] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[57] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International Conference on Machine Learning (ICML)*, 2014, pp. 1278–1286.

[58] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[59] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[60] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[61] J. He, D. Spokoyny, G. Neubig, and T. Berg-Kirkpatrick, "Lagging inference networks and posterior collapse in variational autoencoders," in *International Conference on Learning Representations (ICLR)*, 2019.

[62] J. Lucas, G. Tucker, R. Grosse, and M. Norouzi, "Understanding posterior collapse in generative latent variable models," in *ICLR Workshop*, 2019.

[63] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[64] S. Yeung, A. Kannan, Y. Dauphin, and L. Fei-Fei, "Tackling over-pruning in variational autoencoders," in *ICML Workshop*, 2017.

[65] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, Lille, France, 2015, pp. 448–456.

[66] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," in *Conference on Computational Natural Language Learning (CoNLL)*, 2016.

[67] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations (ICLR)*, 2017.

[68] Q.-T. Truong, A. Salah, and H. W. Lauw, "Bilateral variational autoencoder for collaborative filtering," in *ACM International Conference on Web Search and Data Mining*, 2021, pp. 292–300.

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[71] R. M. Shah and V. Kumar, "RRL: Resnet as representation for reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2021, pp. 9465–9476.

[72] C. Wang, X. Luo, K. Ross, and D. Li, "VRL3: A data-driven framework for visual deep reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[73] Z. Yuan, Z. Xue, B. Yuan, X. Wang, Y. Wu, Y. Gao, and H. Xu, "Pre-trained image encoder for generalizable visual reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[74] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine, "Bridge data: Boosting generalization of robotic skills with cross-domain datasets," in *Robotics: Science and Systems (RSS)*, 2022.

[75] A. Khandelwal, L. Weihs, R. Mottaghi, and A. Kembhavi, "Simple but effective: Clip embeddings for embodied AI," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 14 829–14 838.

[76] J. Pari, N. M. M. Shafiullah, S. P. Arunachalam, and L. Pinto, "The surprising effectiveness of representation learning for visual imitation," in *Robotics: Science and Systems (RSS)*, 2022.

[77] I. Radosavovic, T. Xiao, S. James, P. Abbeel, J. Malik, and T. Darrell, "Real-world robot learning with masked visual pre-training," in *Conference on Robot Learning (CoRL)*, 2023, pp. 416–426.

[78] L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T.-Y. Lin, "Learning to see before learning to act: Visual pre-training for manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7286–7293.

[79] S. Parisi, A. Rajeswaran, S. Purushwalkam, and A. Gupta, "The unsurprising effectiveness of pre-trained vision models for control," in *International Conference on Machine Learning (ICML)*, 2022, pp. 17 359–17 371.

[80] A. Foong, W. Bruinsma, D. Burt, and R. Turner, "How tight can PAC-Bayes be in the small data regime?" in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[81] C. Shalizi and A. Kontorovich, "Predictive PAC learning and process decompositions," in *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[82] S. Zhou, Y. Lei, and A. Kabán, "Toward better PAC-Bayes bounds for uniformly stable algorithms," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 29 602–29 614.

[83] B.-E. Chérief-Abdellatif, Y. Shi, A. Doucet, and B. Guedj, "On PAC-Bayesian reconstruction guarantees for VAEs," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022, pp. 3066–3079.

[84] S. D. Mbacke, F. Clerc, and P. Germain, "Statistical guarantees for variational autoencoders using PAC-Bayesian theory," in *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[85] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT press, 2018.

[86] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC model-free reinforcement learning," *International Conference on Machine Learning (ICML)*, pp. 881–888, 2006.

[87] S. M. Kakade, "On the sample complexity of reinforcement learning," Ph.D. dissertation, University College London (United Kingdom), 2003.

[88] A. L. Strehl, L. Li, and M. L. Littman, "Reinforcement learning in finite MDPs: PAC analysis." *Journal of Machine Learning Research*, vol. 10, no. 11, 2009.

[89] O. Catoni, "A PAC-Bayesian approach to adaptive classification," *Technical report, Laboratoire de Probabilités et Modèles Aléatoires*, 2003.

[90] D. A. McAllester, "Some PAC-Bayesian theorems," in *Annual Conference on Computational Learning Theory*, 1998, pp. 230–234.

[91] A. Müller, "Integral probability metrics and their generating classes of functions," *Advances in applied probability*, vol. 29, no. 2, pp. 429–443, 1997.

[92] V. Feldman, "On the power of membership queries in agnostic learning," *Journal of Machine Learning Research*, vol. 10, 2009.

[93] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine learning*, vol. 15, pp. 201–221, 1994.

[94] S. Tong and D. Koller, "Active learning for parameter estimation in Bayesian networks," in *Advances in Neural Information Processing Systems*, vol. 13, 2000.

[95] ——, "Active learning for structure in Bayesian networks," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 2, 2001, pp. 863–869.

[96] T. Huang, P. Chen, and R. Li, "A semi-supervised vae based active anomaly detection framework in multivariate time series for online systems," in *ACM Web Conference*, 2022, pp. 1797–1806.

[97] S. Hwang, J. Choi, and J. Choi, "Uncertainty-based selective clustering for active learning," *IEEE Access*, vol. 10, pp. 110 983–110 991, 2022.

[98] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International Conference on Machine Learning (ICML)*, 2020, pp. 1597–1607.

[99] N. Hansen, H. Su, and X. Wang, "Stabilizing deep q-learning with convnets and vision transformers under data augmentation," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[100] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[101] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, 2019.

[102] X. Liu, Y. Zou, L. Kong, Z. Diao, J. Yan, J. Wang, S. Li, P. Jia, and J. You, "Data augmentation via latent space interpolation for image classification," in *International Conference on Pattern Recognition (ICPR)*, 2018, pp. 728–733.

[103] S. J. Bang, M. J. Kang, M.-G. Lee, and S. M. Lee, "Sto-cvae: state transition-oriented conditional variational autoencoder for data augmentation in disability classification," *Complex & Intelligent Systems*, vol. 10, no. 3, pp. 4201–4222, 2024.

[104] C. Cai and S. Ferrari, "Information-driven sensor path planning by approximate cell decomposition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 39, no. 3, pp. 672–689, 2009.

[105] D. Fox, W. Burgard, and S. Thrun, "Active Markov localization for mobile robots," *Robotics and Autonomous Systems*, vol. 25, no. 3-4, pp. 195–207, 1998.

[106] C. Kreucher, K. Kastella, and A. O. Hero, "Sensor management using an active sensing approach," *Signal Processing*, vol. 85, no. 3, pp. 607–624, 2005.

[107] J. Cooper and M. Goodrich, "Towards combining UAV and sensor operator roles in UAV-enabled visual search," in *IEEE International Conference on Human Robot Interaction (HRI)*, 2008, pp. 351–358.

[108] J. Toh and S. Sukkarieh, "A Bayesian formulation for the prioritized search of moving objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006, pp. 219–224.

[109] G. A. Hollinger, B. Englot, F. S. Hover, U. Mitra, and G. S. Sukhatme, "Active planning for underwater inspection and the benefit of adaptivity," *International Journal of Robotics Research*, vol. 32, no. 1, pp. 3–18, 2013.

[110] T. Arbel and F. Ferrie, "Viewpoint selection by navigation through entropy maps," in *IEEE International Conference on Computer Vision*, vol. 1, 1999, pp. 248–254.

[111] J. Denzler and C. Brown, "Information theoretic sensor data selection for active object recognition and state estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 145–157, 2002.

[112] Y. Ye and J. K. Tsotsos, "Sensor planning for 3D object search," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 145–168, 1999.

[113] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, "Viewpoint selection using viewpoint entropy." in *Vision Modeling and Visualization Conference*, vol. 1, 2001, pp. 273–280.

[114] N. A. Massios and R. B. Fisher, "A best next view selection algorithm incorporating a quality criterion," in *British Machine Vision Conference*, 1998.

[115] Y. Takeuchi, N. Ohnishi, and N. Sugie, "Active vision system based on information theory," *Systems and Computers in Japan*, vol. 29, no. 11, pp. 31–39, 1998.

[116] S. Chen, L. Zheng, Y. Zhang, Z. Sun, and K. Xu, "VERAM: View-enhanced recurrent attention model for 3D shape classification," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 12, pp. 3244–3257, 2018.

[117] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1912–1920.

[118] E. Johns, S. Leutenegger, and A. J. Davison, "Pairwise decomposition of image sequences for active multi-view recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3813–3822.

[119] A. Bender, S. B. Williams, and O. Pizarro, "Autonomous exploration of large-scale benthic environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 390–396.

[120] N. Cao, K. H. Low, and J. M. Dolan, "Multi-robot informative path planning for active sensing of environmental phenomena: A tale of two algorithms," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013, pp. 7–14.

[121] R. Marchant and F. Ramos, "Bayesian optimisation for informative continuous path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6136–6143.

[122] C. Leung, S. Huang, N. Kwok, and G. Dissanayake, "Planning under uncertainty using model predictive control for information gathering," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 898–910, November 2006.

[123] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous UAV path planning and estimation," *IEEE Robotics and Automation Magazine*, vol. 16, no. 2, pp. 35–42, June 2009.

[124] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, vol. 34, pp. 707–755, 2009.

[125] N. Atanasov, B. Sankaran, J. Le Ny, G. Pappas, and K. Daniilidis, "Nonmyopic view planning for active object classification and pose estimation," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1078–1090, Oct 2014.

[126] H. J. S. Feder, J. J. Leonard, and C. M. Smith, "Adaptive mobile robot navigation and mapping," *International Journal of Robotics Research*, vol. 18, no. 7, pp. 650–668, 1999.

[127] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas, "Cooperative air and ground surveillance," *IEEE Robotics and Automation Magazine*, vol. 13, no. 3, pp. 16–25, 2006.

[128] C. Kreucher, J. Wegrzyn, M. Beauvais, and R. Conti, "Multiplatform information-based sensor management: an inverted UAV demonstration," in *SPIE Defense Transformation and Network-Centric Systems*, vol. 6578, 2007, pp. 304–314.

[129] W. Lu, G. Zhang, S. Ferrari, R. Fierro, and I. Palunko, "An information potential approach for tracking and surveilling multiple moving targets using mobile sensor agents," in *SPIE Unmanned Systems Technology*, vol. 8045, 2011, pp. 267–279.

[130] F. Bourgault, A. A. Makarenko, S. Williams, B. Grocholsky, and H. Durrant-Whyte, "Information based adaptive robotic exploration," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, 2002, pp. 540–545.

[131] Y. F. Li and Z. G. Liu, "Information entropy based viewpoint planning for 3-D object reconstruction," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 324–327, 2005.

[132] X. Liao and L. Carin, "Application of the theory of optimal experiments to adaptive electromagnetic-induction sensing of buried targets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 8, pp. 961–972, 2004.

[133] E.-M. Wong, F. Bourgault, and T. Furukawa, "Multi-vehicle Bayesian search for multiple lost targets," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 3169–3174.

[134] J. Vander Hook, P. Tokekar, and V. Isler, "Cautious greedy strategy for bearing-based active localization: Experiments and theoretical analysis," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 1787–1792.

[135] M. Rahimi, M. Hansen, W. Kaiser, G. Sukhatme, and D. Estrin, "Adaptive sampling for environmental field estimation using robotic sensors," in *IEEE International Conference on Intelligent Robots and Systems*, Aug 2005, pp. 3692–3698.

[136] C. Stachniss and W. Burgard, "Exploring unknown environments with mobile robots using coverage maps," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 1127–1134.

[137] K. H. Low, J. M. Dolan, and P. Khosla, "Adaptive multi-robot wide-area exploration and mapping," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008, pp. 23–30.

[138] G. Mathew and I. Mezić, "Metrics for ergodicity and design of ergodic dynamics for multi-agent systems," *Physica D: Nonlinear Phenomena*, vol. 240, no. 4, pp. 432–442, 2011.

[139] L. Miller, Y. Silverman, M. A. MacIver, and T. Murphey, "Ergodic exploration of distributed information," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 36–52, 2016.

[140] A. Mavrommati, E. Tzorakoleftherakis, I. Abraham, and T. D. Murphey, "Real-time area coverage and target localization using receding-horizon ergodic exploration," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 62–80, 2018.

[141] I. Abraham and T. Murphey, "Decentralized ergodic control: Distribution-driven sensing and exploration for multi-agent systems," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2987–2994, 2018.

[142] K. Fitzsimons, A. M. Acosta, J. Dewald, and T. D. Murphey, "Ergodicity reveals assistance and learning in physical human robot interaction," in *Robotics: Science and Systems (RSS)*, vol. 4, no. 29, 2019.

[143] C. Chen, T. D. Murphey, and M. A. MacIver, "Tuning movement for sensing in an uncertain world," *eLife*, 2020.

[144] I. Abraham, A. Prabhakar, and T. D. Murphey, "An ergodic measure for active learning from equilibrium," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 917–931, 2021.

[145] L. M. Miller, Y. Silverman, M. A. MacIver, and T. D. Murphey, "Ergodic exploration of distributed information," *IEEE Transactions on Robotics*, vol. 32, pp. 36–52, 2016.

[146] A. Mavrommati, E. Tzorakoleftherakis, I. Abraham, and T. D. Murphey, "Real-time area coverage and target localization using receding-horizon ergodic exploration," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 62–80, 2018.

[147] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller, "Deepmind control suite," *arXiv*, 2018.

[148] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu based physics simulation for robot learning," in *Advances in Neural Information Processing Systems*, vol. 35, 2021.

[149] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra1, "Habitat: A platform for embodied AI research," in *IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 9339–9347.

[150] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Robotics: Science and Systems (RSS)*, 2018.

[151] M. Torne, A. Simeonov, Z. Li, A. Chan, T. Chen, A. Gupta, and P. Agrawal, "Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation," in *Robotics: Science and Systems (RSS)*, 2024.

[152] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," in *Robotics: Science and Systems (RSS)*, 2024.

[153] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[154] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Conference on Robot Learning (CoRL)*, 2018.

[155] J. Leitner, A. W. Tow, N. Sünderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool, P. T. Kujala, L. Nicholson, T. Pham, J. Sergeant, L. Wu, F. Zhang, B. Upcroft, and P. Corke, "The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[156] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[157] N. Gürtler, S. Blaes, P. Kolev, F. Widmaier, M. Wuthrich, S. Bauer, B. Schölkopf, and G. Martius, "Benchmarking offline reinforcement learning on real-robot hardware," in *International Conference on Learning Representations (ICLR)*, 2023.

[158] B. Yang, D. Jayaraman, J. Zhang, and S. Levine, "REPLAB: A reproducible low-cost arm benchmark for robotic learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[159] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar, "Robel: Robotics benchmarks for learning with low-cost robots," in *Conference on Robot Learning (CoRL)*, 2019.

[160] H. Axelsson, Y. Wardi, M. Egerstedt, and E. Verriest, "Gradient descent approach to optimal mode scheduling in hybrid dynamical systems," *Journal of Optimization Theory and Applications*, vol. 136, no. 2, pp. 167–186, 2008.

[161] R. Vasudevan, H. Gonzalez, R. Bajcsy, and S. S. Sastry, "Consistent approximations for the optimal control of constrained switched systems—part 1: A conceptual algorithm," *SIAM Journal on Control and Optimization*, vol. 51, no. 6, pp. 4463–4483, 2013.

[162] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, 2000.

[163] E. A. Theodorou and E. Todorov, "Relative entropy and free energy dualities: Connections to path integral and KL control," in *IEEE International Conference on Decision and Control (CDC)*, 2012, pp. 1466–1473.

[164] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1433–1440.

[165] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. Rehg, and E. A. Theodorou, "Robust sampling based model predictive control with sparse objective information," in *Robotics: Science and Systems (RSS)*, 2018.

[166] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI Conference on Artificial Intelligence*, vol. 8, 2008, pp. 1433–1438.

[167] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, "Modeling interaction via the principle of maximum causal entropy," in *International Conference on Machine Learning (ICML)*, 2010, pp. 1255–1262.

[168] E. Todorov, "Efficient computation of optimal actions," *Proceedings of the National Academy of Sciences*, vol. 106, no. 28, pp. 11 478–11 483, 2009.

[169] M. Toussaint, "Robot trajectory optimization using approximate inference," in *International Conference on Machine Learning (ICML)*, 2009, pp. 1049—1056.

[170] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," in *Robotics: Science and Systems (RSS)*, 2012, pp. 353–361.

[171] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning (ICML)*, 2013.

[172] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *International Conference on Machine Learning (ICML)*, vol. 70, 2017, pp. 1352–1361.

[173] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Robotics: Science and Systems (RSS)*, 2018.

[174] S. B. Thrun, "Efficient exploration in reinforcement learning," Carnegie Mellon University, Tech. Rep., 1992.

[175] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup, "A survey of exploration methods in reinforcement learning," *arXiv*, 2021.

[176] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer, 2013, vol. 6.

[177] J. P. Hespanha, *Linear Systems Theory: Second Edition*. Princeton University Press, 2018.

[178] E. Todorov, "Linearly-solvable Markov decision problems," in *Advances in Neural Information Processing Systems*, vol. 19, 2007.

[179] D. Mitra, "W matrix and the geometry of model equivalence and reduction," *Proceedings of the Institution of Electrical Engineers*, vol. 116, pp. 1101–1106, 1969.

[180] S. Dean, H. Mania, N. Matni, B. Recht, and S. Tu, "On the sample complexity of the linear quadratic regulator," *Foundations of Computational Mathematics*, vol. 20, no. 4, pp. 633–679, 2020.

[181] A. Tsiamis and G. J. Pappas, "Linear systems can be hard to learn," *IEEE International Conference on Decision and Control (CDC)*, pp. 2903–2910, 2021.

[182] A. Tsiamis, I. M. Ziemann, M. Morari, N. Matni, and G. J. Pappas, "Learning to control linear systems can be hard," in *Conference on Learning Theory (COLT)*, vol. 178, 2022, pp. 3820–3857.

[183] E. T. Jaynes, "Information theory and statistical mechanics," *Phys. Rev.*, vol. 106, pp. 620–630, 1957.

[184] P. D. Dixit, J. Wagoner, C. Weistuch, S. Pressé, K. Ghosh, and K. A. Dill, "Perspective: Maximum caliber is a general variational principle for dynamical systems," *The Journal of Chemical Physics*, vol. 148, no. 1, 2018.

[185] P. Chvykov, T. A. Berrueta, A. Vardhan, W. Savoie, A. Samland, T. D. Murphey, K. Wiesenfeld, D. I. Goldman, and J. L. England, "Low rattling: A predictive principle for self-organization in active collectives," *Science*, vol. 371, no. 6524, pp. 90–95, 2021.

[186] J. N. Kapur, *Maximum Entropy Models in Science and Engineering*. John Wiley & Sons, 1989.

[187] M. Kardar, *Statistical Physics of Fields*. Cambridge University Press, 2007.

[188] M. Hairer, "Lecture notes on ergodic properties of Markov chains," 7 2018.

[189] C. C. Moore, "Ergodic theorem, ergodic theory, and statistical mechanics," *Proceedings of the National Academy of Sciences*, vol. 112, no. 7, pp. 1907–1911, 2015.

[190] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, vol. 80, 2018, pp. 1861–1870.

[191] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, ser. Probability and Statistics. John Wiley & Sons, 2014.

[192] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," in *International Conference on Machine Learning (ICML)*, 2017.

[193] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI Conference on Artificial Intelligence*, vol. 32, no. 392, 2018.

[194] J. Moos, K. Hansel, H. Abdulsamad, S. Stark, D. Clever, and J. Peters, "Robust reinforcement learning: A review of foundations and recent advances," *Machine Learning and Knowledge Extraction*, vol. 4, no. 1, pp. 276–315, 2022.

[195] X. Wang, W. Deng, and Y. Chen, "Ergodic properties of heterogeneous diffusion processes in a potential well," *The Journal of Chemical Physics*, vol. 150, no. 16, p. 164121, 2019.

[196] R. G. Palmer, "Broken ergodicity," *Advances in Physics*, vol. 31, no. 6, pp. 669–735, 1982.

[197] J. W. Krakauer, A. M. Hadjiosif, J. Xu, A. L. Wong, and A. M. Haith, *Motor Learning*. John Wiley & Sons, 2019, pp. 613–663.

[198] A. Ames, J. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *IEEE International Conference on Decision and Control (CDC)*, 2014.

[199] A. Taylor, A. Singletary, Y. Yue, and A. Ames, "Learning for safety-critical control with control barrier functions," in *Conference on Learning for Dynamics and Control (L4DC)*, vol. 120, 2020, pp. 708–717.

[200] W. Xiao, T.-H. Wang, R. Hasani, M. Chahine, A. Amini, X. Li, and D. Rus, "Barriernet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics*, 2023.

[201] A. T. Taylor, T. A. Berrueta, and T. D. Murphey, "Active learning in robotics: A review of control principles," *Mechatronics*, vol. 77, 2021.

[202] H. S. Seung, H. Sompolinsky, and N. Tishby, "Statistical mechanics of learning from examples," *Phys. Rev. A*, vol. 45, pp. 6056–6091, 1992.

[203] B. J. Van Stratum, M. P. Austin, K. Shoele, and J. E. Clark, "Comparative model evaluation with a symmetric three-link swimming robot," in *IEEE International Conference on Intelligent Robots and Systems*, 2022.

[204] R. Hall, G. Espinosa, S.-S. Chiang, and C. D. Onal, "Design and testing of a multimodule, tetherless, soft robotic eel," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[205] S. Hirose and A. Morishima, "Design and control of a mobile robot with an articulated body," *International Journal of Robotics Research*, vol. 9, no. 2, pp. 99–114, 1990.

[206] C. Wright, A. Johnson, A. Peck, Z. McCord, A. Naaktgeboren, P. Gianfortoni, M. Gonzalez-Rivero, R. Hatton, and H. Choset, "Design of a modular snake robot," in *IEEE International Conference on Intelligent Robots and Systems*, 2007.

[207] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv*, 2014.

[208] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[209] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S.-Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H.-C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[210] W. Gao, X.-Y. Niu, and Z.-H. Zhou, "Learnability of non-iid," in *Asian Conference on Machine Learning*, 2016, pp. 158–173.

[211] M. Mohri and A. Rostamizadeh, "Stability bounds for non-iid processes," in *Advances in Neural Information Processing Systems*, vol. 20, 2007.

[212] L. Zhang and L. Guo, "Asymptotically efficient online learning for censored regression models under non-iid data," *arXiv*, 2023.

[213] C. R. Givens and R. M. Shortt, "A class of Wasserstein metrics for probability distributions," *Michigan Mathematical Journal*, vol. 31, no. 2, pp. 231–240, 1984.

[214] M. Sun, A. Gaggar, P. Trautman, and T. Murphey, "Fast ergodic search with kernel functions," *arXiv*, 2024.

[215] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[216] M. D. Lo, S. H. Ackley, and P. Solari, "Homemade ultrasound phantom for teaching identification of superficial soft tissue abscess," *Emergency Medicine Journal*, vol. 29, no. 9, pp. 738–741, 2012.

[217] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[218] OpenAI, "Roboschool, open-source software for robot simulation, integrated with openai gym," *GitHub repository*, 2017.

[219] J. A. Boyan, "Least-squares temporal difference learning," in *International Conference on Machine Learning (ICML)*, 1999.

[220] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *International Conference on Machine Learning (ICML)*, 2001.

[221] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[222] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009.

[223] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[224] Intel®, "oneCCL bindings for PyTorch," *GitHub repository*, 2020.

Appendix A

Supplementary Material for Chapter 3

A.1 General Implementation Details

All simulated examples use the reward functions specified in the Pybullet [**217**], Roboschool [**218**], or MuJoCo environments [**13**] unless otherwise specified. Table A–1 lists the hyperparameters used for each environment. Any parameter not explicitly mentioned as deterministic or stochastic formulations of hybrid learning are equivalent.

A.1.1 Simulation Benchmarks

The goal for both the cartpole and Acrobot environments is to swing the pendulum up to the vertical position. We refer to both of these examples as "swing-up tasks". We chose these tasks to demonstrate the performance capabilities of the model-based method. Pendulum swing-up tasks are well understood, low dimension, underactuated control problems for which we anticipated the model-based method would out-perform the model-free method. Given enough samples, the model-free method should also be able to learn these tasks, but the model-free method would require many more samples to learn the task.

Environment	Simulator	H	T	K	λ	Policy Dim	Learning Rate
Cartpole Swing-up	Roboschool	5	1000	60	0.1	128	0.003
Acrobot Swing-up	Gym	5	500	20	0.1	128	0.003
Hopper	MuJoCo	5	1000	60	0.2	128	0.003
Half-Cheetah	MuJoCo	10	1000	60	0.2	128	0.003
Sawyer (push)	n/a	10	100	10	0.01	128×128	0.0003
Sawyer (clutter)	n/a	10	200	40	0.1	128×128	0.0003

Table A–1 Parameters for all examples used in the paper (only when applicable). Each example using the deterministic formulation of hybrid learning (Alg. 3–1) uses added action exploration of the form $\epsilon = 0.999^t$ where t is the total number of environment interactions.

We modified the Gym Acrobot environment [**12**] to test continuous states and updated the reward function to include dependencies on actions and states. The reward function is

$$r = -0.001a^2 - \cos(\theta_1) - \cos(\theta_1 + \theta_2) \tag{A.1}$$

where a, θ_1, θ_2 are the control, the shoulder angle, and the (relative) elbow angle respectively.

The goal for both the hopper environment and the half cheetah environments is to learn a gait to maximize velocity in the $+x$ direction in a 2D environment. We refer to both of these examples as "locomotion tasks". We chose the two locomotion tasks to demonstrate the advantages of model-free methods. Model-free methods have been used to learn control tasks in high-dimensional spaces, so we anticipated the model-free method would out-perform the model-based methods for the locomotion tasks.

A.1.2 Models

For each deterministic simulated benchmark example, we use the model-predictive controller from [**1**]. For all other experiments, we use Neural-Network Model Predictive Path Integral Control (NN-MPPI) [**2**]. For all model-based and hybrid learning simulated examples, the dynamics are represented by $s_{t+1} = s_t + f(s_t, a_t)$, where $f(s_t, a_t) = \mathbf{W}_2\ \sigma(\mathbf{W}_1[s_t, a_t] + \mathbf{b}_1) + \mathbf{b}_2$. The transition function $f(s_t, a_t)$ is modeled as a single layer neural network with

128 hidden nodes. For all tasks, we use the Rectifying Linear Unit (ReLU) nonlinearity. $\mathbf{W}_1 \in \mathbb{R}^{200\times(n+m)}$, $\mathbf{W}_2 \in \mathbb{R}^{n\times 200}$, $\mathbf{b}_1 \in \mathbb{R}^{200}$, $\mathbf{b}_2 \in \mathbb{R}^n$ are learned parameters. The reward function is modeled as a two layer network with 200 hidden nodes and the ReLU nonlinearity. The reward function and dynamics model are optimized using Adam [**207**] with learning rates specified in Table A–1. The model is regularized using the negative log-loss of a normal distribution where the variance, $\Sigma_{\text{model}} \in \mathbb{R}^{n\times n}$, is a hyperparameter that is simultaneously learned based on experience. The predicted reward utility is improved by the error between the predicted target and target reward equal to $\mathcal{L} = \|r_t + 0.95\ r(s_{t+1}, a_{t+1}) - r(s_t, a_t)\|^2$. The structure of this loss function is similar to temporal-difference learning [**219**, **220**]. The reward term with the next state and next action helps the models learn in environments with rewards that do not strictly depend on the current state, as with some MuJoCo locomotion tasks. We use a batch size of 128 samples.

A.1.3 Policies

For the model-free and hybrid learning examples, we use Soft Actor Critic (SAC) to update our model-free policy. For the Sawyer robot push experiments, we use the SAC implementation in [**11**]. For all other experiments, we use the SAC implementation in [**3**]. We use the hyperparameters specified by the shared parameters in [**3**,**11**] including the structure of the soft Q functions and automatic gradient-based temperature tuning method and excluding the batch size and policy. Instead, we match the batch size used with model learning and use a simpler policy representation. Our policy is parameterized by a normal distribution with a mean function defined as a single layer network with the ReLU nonlinearity and 128 hidden nodes. The diagonal of the variance is also specified using a single layer network

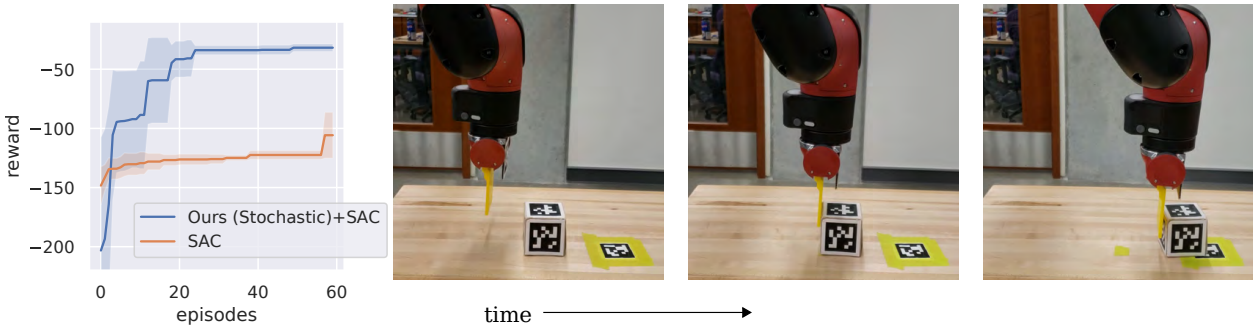

Figure A–1 Hybrid learning Sawyer push experiment results on the Sawyer robot (averaged over 5 trials). The task is to push a block to a designated target through environment interactions (see time-series results above). Our method is able achieve the task within 3 minutes (each episode takes 10 seconds) through effectively using both predictive models and model-free methods. The same amount of interaction with SAC was unable to successfully push the block to the target.

with 128 hidden nodes and a ReLU nonlinearity. The SAC parameters are held constant across experiments to remove the impact of hyperparameter tuning.

A.1.4 Robot Experiments

For hardware experiments, a camera is used to identify the location of objects in the environment using landmark tags and color image processing. For the Sawyer robot push example, the state is defined as the pose of the end-effector of the arm relative to the block as well as the pose of the block relative to the target (state dimension is 4). The action space is the target end-effector velocity (action dimension is 2). The reward is defined as

$$r(s, a) = -5\|p_{\text{b2t}}\| - 10\|p_{\text{ee2b}}\| - 0.01\|a\|^2$$

where $p_{\text{b2t}}, p_{\text{ee2b}}$ denote the poses of the block to the target and the end-effector to the target location respectively. For this experiment, the goal is to push a block on a table to a specified marker. The position of the marker and the block are known to the robot. The robot is rewarded for pushing the block to the marker. What makes this task difficult is that the robot needs to discover contact between the arm and the block to complete the pushing task.

Shown in Fig. A–1 our hybrid learning approach is able to learn the task within 20 episodes (total time is 3 minutes, 10 seconds for each episode). Since our method naturally relies on the predictive models when the policy is uncertain, the robot is able to plan through the contact to achieve the task whereas SAC takes significantly longer to discover the pushing dynamics. As a result, we are able to achieve the task with minimal environment interaction.

For the Sawyer robot clutter example, the robot has no knowledge of the physical properties of the blocks (including size, weight, and material) or the true location of objects in the world. The state is comprised of the position of each of the six blocks in the workspace relative to the robot's end-effector (state dimension is 12). The "target block" is always the a fixed location in the state vector, but the five "clutter" blocks are sorted from closest to furthest from the end-effector to enable generalization. Without this sorting, the experimenter would also need to shuffle the "clutter" blocks around the "target" block, which would extend the testing duration. The action space is the robot's end-effector velocity (action dimension is 2). The reward is defined as

$$r(s, a) = -\|p_{\text{ee2t}}\| - \|p_{\delta \text{t}}\| - 0.01\|a\|^2 \tag{A.2}$$

where $p_{\text{ee2t}}, p_{\delta \text{t}},$ and a are the pose of the target block relative to the end-effector, the change in target block position relative to the prior time step, and the control respectively. Each trial can be a maximum of 200 steps, but due to physical workspace constraints, we terminate the episode if the robot pushes the target block outside of the workspace. To account for this early termination, we post-processed the data to add a penalty proportional to the number of episode steps remaining when the block was pushed out of reach.

Algorithm	Pretrain	H	Model-Free Method	Model Dim	Policy Dim	Learning Rate
MVE	1000	3	DDPG	200×200	128	0.003
STEVE	1000	3	DDPG	200×200	128	0.003
MBPO	1000	3	SAC	200×200	128	0.003
GPS	n/a	n/a	n/a	n/a (GMM)	128	0.003
MB-MF	1000	3	PPO, TRPO	200×200	128	0.003

Table A–2 Parameters for related work comparisons (when applicable).

A.2 Related Work Implementation Details

This section details the modifications made to related work implementations to compare to the hybrid results presented in this paper. Table A–2 provides a lists the general hyperparameters used for each related work algorithm tested. Hyperparameters specific to each algorithms are specified below. Several algorithms assumed access to or learned a termination function. For these comparisons, we excluded the termination function from all implementations. Many of the comparison methods required pre-training their models, so we added 1000 steps of pre-training with random actions to our stochastic hybrid learning algorithm. We also increased the size of our neural network model to 2 layers for these comparisons. Each comparison algorithm is described below.[1]

A.2.1 Model-Based Value Expansion (MVE)

MVE uses only the model-free policy to interact with the environment. Experience is used to fit a dynamics model. During each update, the model is used to imagine future transitions for rollouts of fixed short duration horizons (H). These imagined transitions are incorporated into the Q-value target estimation (critic) update. The original implementation only learned a dynamics transition function and assumed access to a reward function and a

[1] These plots are breakouts of the results presented in Fig. 3–5. Markers are intended to distinguish between methods and do not represent data points.

termination function [**4**]. We removed the termination function and added a learned reward function. Fig. A–2 shows the performance curves for hybrid learning and MVE.

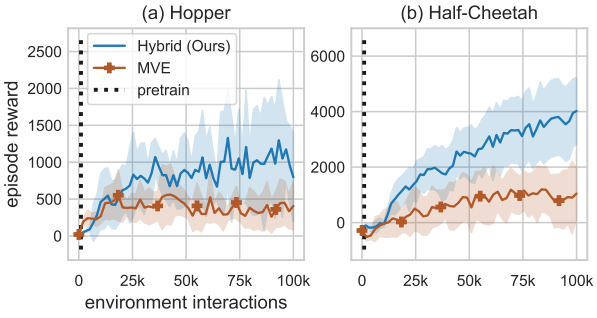

Figure A–2 Pairwise comparison to MVE

Hyperparameter	Value
explore chance	0.05
model updates per epoch	1000
policy updates per epoch	1000
environment steps per epoch	250

Table A–3 MVE hyperparameters

A.2.2 Stochastic Ensemble Value Expansion (STEVE)

STEVE builds on MVE by performing rollouts on all horizon lengths between 0 and H. This algorithm additionally adds the ability to learn an ensemble of models. The original implementation learned a dynamics transition function, a reward function, and a termination function [**5**]. We removed the termination function and tested only a single model for these comparisons. Fig. A–3 shows the performance curves for hybrid learning and STEVE.

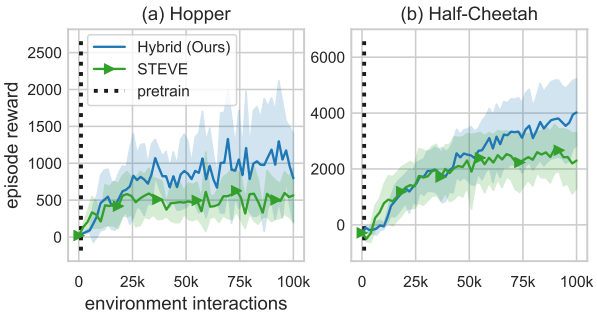

Figure A–3 Pairwise comparison to STEVE

Hyperparameter	Value
ensemble size	1
explore chance	0.05
model updates per epoch	1000
policy updates per epoch	1000
environment steps per epoch	250

Table A–4 STEVE hyperparameters

A.2.3 Model-Based Policy Optimization (MBPO)

MBPO optimizes a policy under a learned model [**6**]. Similar to MVE and STEVE, only the policy interacts with the environment and collects data. The original implementation

assumed the termination function was known and used an ensemble of dynamics models. Additionally, the original implementation used all data collected from the environment to train the models during each update model update iteration. We removed the termination function and tested only a single model for these comparisons. We also used a subset of the collected data for model updates. Fig. A–4 shows the performance curves for hybrid learning and MBPO.

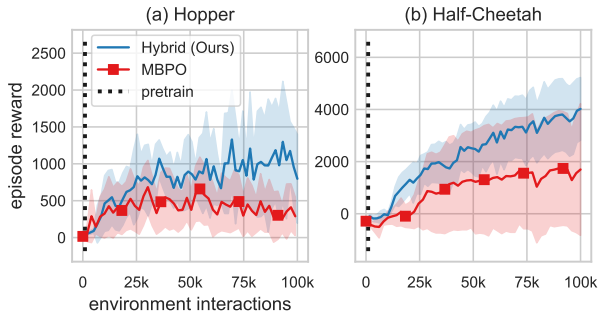

Figure A–4 Pairwise comparison to MBPO

Hyperparameter	Value
ensemble size	1
model train frequency	250
model train batch size	10,000
model rollouts per policy update	20
policy updates per epoch	20,000
environment steps per epoch	1000

Table A–5 MBPO hyperparameters

A.2.4 Guided Policy Search (GPS)

GPS interacts with the environment via the model-based controller rather than the model-free policy. The dynamics are modeled to be Gaussian-linear time varying and the reward function is assumed to be known. The dynamics are modeled as a Gaussian mixture model (GMM), and the algorithm assumes the reward function is known. The policy is trained via behavior cloning with constraints preventing the policy updates from deviating too far from the last implemented trajectory. The original implementations of GPS often used shorter horizon tasks than the full gym benchmarks, but we used the original length episodes for comparison purposes. We used the Mirror Descent variant of Guided Policy Search (MD-GPS) [**7**]. Another difference between this simulation and the others is that it cannot handle true early termination due to the time varying controller. The gym hopper

environment "ends" when the hopper falls out of a desired configuration. For GPS testing of the gym hopper environment, the episode termination output is ignored and instead the "alive bonus" (typically one in the gym environment) is toggled to zero when done is called by the simulator. Fig. A–5 shows the performance curves for hybrid learning and GPS.

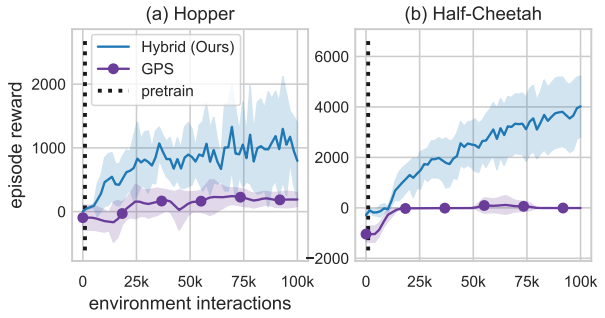

Figure A–5 Pairwise comparison to GPS

Hyperparameter	Value
timesteps per iteration	5000
kl step	1
dynamics GMM clusters	20
policy GMM clusters	20

Table A–6 GPS hyperparameters

A.2.5 Model-Free Model-Based (MB-MF)

MB-MF uses a multi-stage approach [**8**]. First, a controller is learned using a random shooting method. Then the controller is transferred to a model-free policy using data aggregation. Finally, the policy is fine-tuned using a model-free method. The original method presented in the paper was able to fully observe all environment states, but the version implemented here only had access to the default gym environment (partially observable) state. Fig. A–6 shows the performance curves for hybrid learning and MB-MF.

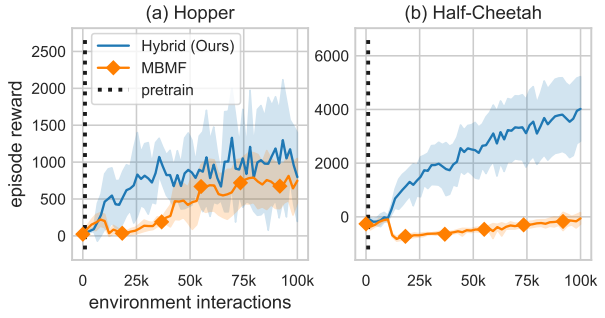

Figure A–6 Pairwise comparison to MB-MF

Hyperparameter	Value
search population size	1000
timesteps per epoch	1000
model based timesteps	10,000
data aggregation epochs	100
data aggregation iterations	7

Table A–7 MB-MF hyperparameters

Appendix B

Supplementary Material for Chapter 4

All simulated examples use the reward functions specified MuJoCo environments unless otherwise specified [**12**, **13**]. Table B–1 provides a list of all hyperparameters used in all implementations of Maximum Diffusion Reinforcement Learning (MaxDiff RL), Neural-Network Model Predictive Path Integral Control (NN-MPPI), and Soft Actor Critic (SAC), for each environment. All experiments were run for a total of 1 million environment steps with each episode including 1000 steps. For multi-shot tests, the episode was reset upon satisfying an episode termination condition or completing the number of steps in an episode. For single-shot tests, the environment was never reset and each episode only constituted a checkpoint for saving cumulative rewards during the duration of that episode. All representations used Rectifying Linear Unit (ReLU) activation functions, and 10 seeds were run for each configuration. For all model-based examples (*i.e.*, MaxDiff RL and NN-MPPI [**2**]), the transition and reward functions are both modeled by fully-connected neural network representation; see the stochastic model description in Section A.1.2 for more details. For all model-free examples, we implement SAC to provide updates to our model-free policy [**11**];

see the policy description in Section A.1.3 for more details. All reinforcement learning experiments were run on an Intel® Xeon(R) Platinum 8380 CPU @ 2.30GHz x 160 server[1] running Ubuntu 18.04 and Python 3.6.

Algorithm	**Hyperparameter**		**2D Point-mass**	**MuJoCo Gym (v3)**		
				Swimmer	**Ant**	**Half-cheetah**
All	State Dim		4	10	29	18
	Action Dim		2	2	8	6
	Learning Rate		0.0005	0.0003	0.0003	0.0003
	Batch Size		128	128	256	256
SAC	Policy Layers		128×128	256×256	$512 \times 512 \times 512$	256×256
	Discount		0.99	0.99	0.99	0.99
	Smoothing coeff.		0.01	0.005	0.005	0.005
	Reward Scale		0.25	100	5	5
NN-MPPI/ MaxDiff RL (Planning)	Model Layers		128×128	200×200	$512 \times 512 \times 512$	200×200
	Horizon		30	40	20	10
	Discount		0.95	0.95	0.95	0.95
	Multi	Samples	500	500	1000	500
		Lambda	0.5	0.5	0.5	0.5
	SS	Samples	NA	1000	1000	1000
		Lambda		0.1	0.5	0.5
MaxDiff RL (Exploration)	Multi	Alpha	5	1,5,10,50, 100,500,1000	15	5
		Dimensions, Weights (diag.)	$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.01 \\ 0.01 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.05 \\ 0.05 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ z \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.005 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.05 \\ 0.05 \end{bmatrix}$
	SS	Alpha	NA	50	15	5
		Dimensions, Weights (diag.)		$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.05 \\ 0.05 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.05 \\ 0.05 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0.05 \\ 0.05 \end{bmatrix}$

Table B–1 Simulation hyperparameters. *Multi* parameters only apply to multi-shot runs, and *SS* parameters only apply to single-shot runs.

[1] This hardware was loaned by the Intel Corporation, whose technical support we acknowledge.

B.1 Point-mass Environment

The goal of the point-mass environment is to learn to move to the origin of a 2D environment. This is a custom environment in which the point-mass dynamics are simulated as a 2D double integrator with states $[x, y, \dot{x}, \dot{y}]$ and actions $[\ddot{x}, \ddot{y}]$. Each episode is initialized at state $[-1, -1, 0, 0] + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.01)$. The reward function is specified in terms of location in the environment $r = -(x^2 + y^2)$. For multi-shot tests, the episode was terminated if the point-mass exceeded a boundary defined as a square at $x, y = \pm 5$. The simulation uses RK-4 integration with a time step of 0.1.

B.2 MuJoCo Environments

The goal of the following MuJoCo locomotion environments is to learn a gait to maximize velocity in the $+x$ direction by applying joint torques. We use the "v3" environment variants, which include all configuration states in the observation generated at each step.

Swimmer The swimmer operates in a viscous 2D environment, which is described in more detail in Section 5.2. For the "heavy-tailed" tests, the default swimmer model file (`.xml`) is used—a 3-link body with identical links. For the "light-tailed" tests, we modify the density of the "tail" link to be $10\times$ lighter than the other two links. The default link density in the swimmer is 1000 and modified tail density is 100. There is no episode termination condition.

Ant We modify the reward function to zero out the control cost, contact cost, and healthy reward weights, so the reward function only depends on the change in the x-position during the duration of the step (with positive reward for progress in the positive x-direction). We also modified the episode termination condition to make it possible for the ant to recover from falling. The episode termination condition is triggered if the ant has been upside down

for 1 second; the ant is considered "upside down" if the torso angle—which is nominally perpendicular to the ground—exceeds 2.7 radians.

Half-cheetah The half-cheetah is constrained to the 2D vertical plane. There is no episode termination condition, but it is possible for the half-cheetah to get stuck upside-down.

B.3 Statistical Results

Table B–2 shows that across all learning experiments in Section 4.4, differences between MaxDiff RL and comparison methods are statistically significant ($P < 0.05$) according to an unpaired two-sided Welch's t-test implementation in SciPy [**221**], except for one. However, since the Ant environment breaks ergodicity, we do not expect improvements over MaxEnt RL in multi-shot settings.

		NN-MPPI		**SAC**	
Task		P-value	$P < 0.05$	P-value	$P < 0.05$
Point-mass	$\beta = 1$	< 0.001	True	< 0.001	True
	$\beta = 0.1$	< 0.001	True	< 0.001	True
	$\beta = 0.01$	< 0.001	True	< 0.001	True
	$\beta = 0.001$	< 0.001	True	< 0.001	True
Swimmer	Light Multi	< 0.001	True	< 0.001	True
	Light SS (Return)	0.0131	True	0.0034	True
	Light SS (Distance)	< 0.001	True	< 0.001	True
	Heavy Multi	< 0.001	True	< 0.001	True
	Light-to-Heavy Multi	< 0.001	True	< 0.001	True
	Heavy-to-Light Multi	< 0.001	True	< 0.001	True
Ant	Multi	0.8343	False	< 0.001	True
	SS (Return)	0.0154	True	< 0.001	True
	SS (Distance)	< 0.001	True	< 0.001	True
Half-cheetah	Multi	< 0.001	True	< 0.001	True
	SS (Return)	< 0.001	True	< 0.001	True
	SS (Distance)	< 0.001	True	< 0.001	True

Table B–2 Results of statistical comparisons between MaxDiff RL and alternatives. *Multi* and *SS* indicate multi-shot and single-shot experiments respectively. *Multi* significance was evaluated across 100 eval episodes for each final policy. *SS* significance was evaluated in two ways: (1) by terminal windowed return and (2) by terminal distance traveled in reward direction.

Appendix C

Supplementary Material for Chapter 6

C.1 Implementation Details

All experiments were conducted on a Franka Panda robot arm with a Logitech C270 webcam. Communication with the robot was handled by a laptop (Intel® Core i7-8550U CPU @ 1.80GHz x 8) running Ubuntu 18.04 with a Realtime Kernel (version 5.4.138-rt62) and Robot Operating System (ROS) Melodic [**222**]. All other processes were run on a separate server (Intel® Xeon(R) Platinum 8380 CPU @ 2.30GHz x 160) with Ubuntu 20.04 and ROS Noetic. Computers were connected via Ethernet on a local network.

C.1.1 Control Details

For our planar experiments, the robot has access to its end-effector states (x, y, θ), where θ is yaw rotation about the camera-view axis in the plane. We model the end-effector dynamics as a double integrator and assume states are controlled independently. We project states into the range $[-1, 1]$ and use barrier functions to keep the robot within a specified workspace. We use a planning horizon of 10 steps and a planning time step of 0.2s.

C.1.2 Model Details

All networks are implemented in PyTorch v2.0 [**223**]. Table C–1 contains model architecture details. We use the Adam [**207**] optimizer with a learning rate of 0.001, a batch size of 64 images, and 4 distributed trainers. We use the `torch-ccl` backend to enable CPU distributed training [**224**]. Distributed training is not required, but it allows training updates to occur asynchronously from robot control updates. Additional parallelization could be used to further reduce runtime. We allow the distributed trainers to run asynchronously from the robot controller. To ensure the model update rate matches the data collection rate, the training frequency is throttled to 3 model update steps per data collection step.

Key	CNN	*2D Conv. Filters @ Kernel (stride, pad) + Activation*
	MLP	*Fully Connected Output + Activation*
Latent Space Samples		$z_s = \mu_z + \sigma_z \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $z = \mathcal{N}(\mu_z, \text{diag}(\sigma_z)$
Sensor	Input	3x180x180 ▷ *RGB Image*
Encoder	CNN	10@3x3(2,0) + ReLU → 10@3x3(2,0) + ReLU → 20@5x5(3,0)
	Output	20x14x14 ▷ *Encoded Image*
Main	Input	3923 ▷ *Flattened Encoded Image + State* (x, y, θ)
Encoder	MLP	512 + ReLU → 265 + ReLU → 32
	Output	32 ▷ *Latent Space* (z)
Main	Input	19 ▷ *Latent Space Samples + State* (x, y, θ)
Decoder	MLP	256 + ReLU → 512 + ReLU → 3921
	Output	3921 ▷ *Sensor Prediction + Variance*
Sensor	Input	20x14x14 ▷ *UnFlattened Decoder Sensor Prediction*
Decoder	CNN	20@5x5(3,0) + ReLU → 10@3x3(2,0) + ReLU → 10@3x3(2,1)
	Output	3x180x180 ▷ *RGB image*

Table C–1 Conditional Variational Autoencoder (CVAE) Network Architecture

C.2 Original CVAE Loss Function

For this section, the Franka workspace limits, control details, and model training parameters are listed in Tables C–2, C–3, and C–4, respectively. In this section, we use a fixed β ramp to weigh the KL-loss. During active learning, the goal is to learn models with information-rich latent spaces, while the goal of active search is to find sensor data in the test environment to test our learned fingerprints. Therefore, the latent space quality of the active search quality is less important than collecting formative sensor data. By decreasing the size of the images, increasing the batch size, and increasing the beta ramp rate, the active search model converges on the locations of the objects in the test environment quickly, but this comes at the possible cost of a collapsed latent space (which is ok because we don't use the new actively search latent space for identification, only for data collection).

Parameter	**Value**
x	$[\,0.325\text{ m}, 0.625\text{ m}\,]$
y	$[\,-0.15\text{ m}, 0.15\text{ m}\,]$
θ (yaw)	$[\,-0.75\pi, 0.75\pi\,]$

Table C–2 Franka Workspace Parameters

Parameter	**Value**
Time step (dt)	0.1 sec
Horizon (t_H)	10 steps
Sampling Method	Uniform
Number of Target Samples	2000
Control Barrier Weight (x, y, θ)	50
Control Barrier Power (x, y, θ)	4
Control Penalty (R)	$0.5 \cdot \mathbf{I}$
Gaussian Width (Σ)	$0.05 \cdot \mathbf{I}$
Equilibrium Control Policy	$\mathbf{0}$
Acceleration Limits ($\ddot{x}$, $\ddot{y}$, and $\ddot{\theta}$)	$[-2, 2]$

Table C–3 Control Parameters

Parameter	**Learning**	**Search**
Batch Size	64	128
Number of Trainers	4[a]	1
Warmup Steps	1000	100
Warmup Epoch	10	2
Final β	0.05	0.1
Downsample Image	N/A	4x
Projected Workspace Limits (x, y, θ)	$[-1, 1]$	$[-1.33, 1.33]$
Total Exploration[b] Steps	3000	1000 (2-3 obj.) 3000 (6 obj.)
Optimizer	Adam [**207**]	
Learning Rate	1e-3	
Updates per Iteration	5	

Table C–4 Model Training Parameters for Active Learning and Active Search

[a] Distributed
[b] Total amount of training data collected

Next, we show results for the pipeline with 10 different seeds. Since the tests were run on hardware, there were also physical lighting differences in the room for each experiment, which can be seen by comparing the coloring of the background of the images in Figs. C–1, C–2, and C–3. All other parameters were kept fixed between seeds. See the main text for seed 1 results. The results in Figs. C–1 and C–2 correspond to Fig. 6–7 in the main text. Fig. C–3 corresponds to Fig. 6–8 in the main text.

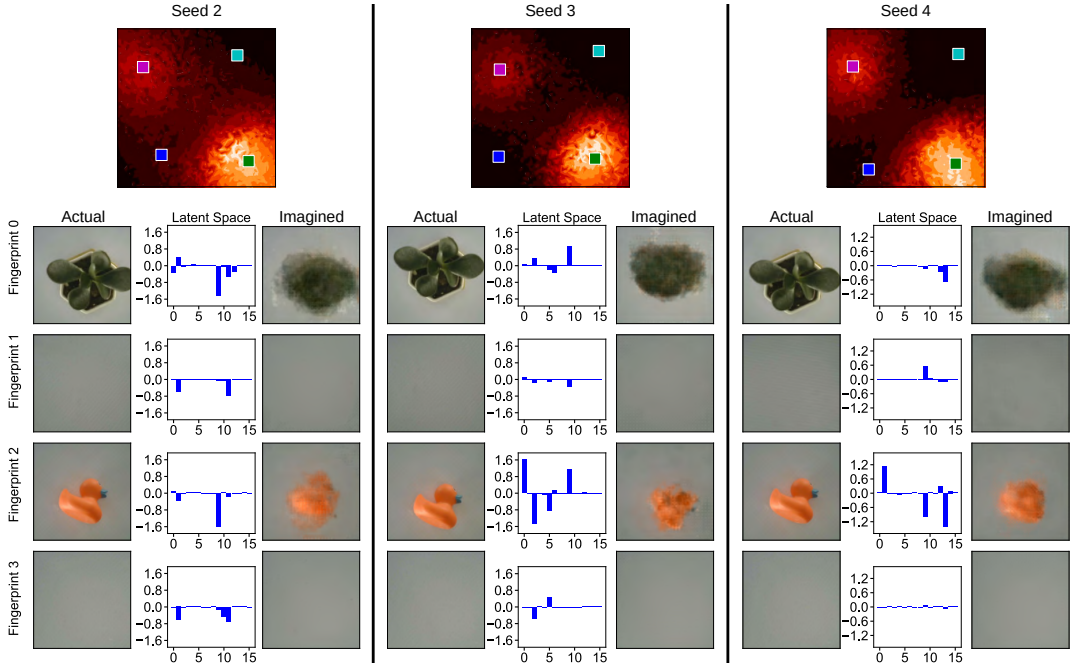

Figure C–1 Fingerprints for Active Learning Seeds 2-4. Each panel displays fingerprint information for actively learned models with different seeds (seeds 5-10 in next figure). All models were trained in the same test workspace (see main paper). The top row shows the locations of the learned cluster means (fingerprint locations). For each fingerprint, a real image, latent space (CVAE encoder output), and imagined image (CVAE decoder output) is included. Qualitatively, the rows from top to bottom can be identified as *plant*, *blank*, *duck*, and *blank* respectively, but the algorithm has no knowledge of these labels. Comparing the panels shows qualitatively that all seeds learned the objects (no latent space collapse), but that the exact workspace belief and imagined images varied between the seeds, as expected.

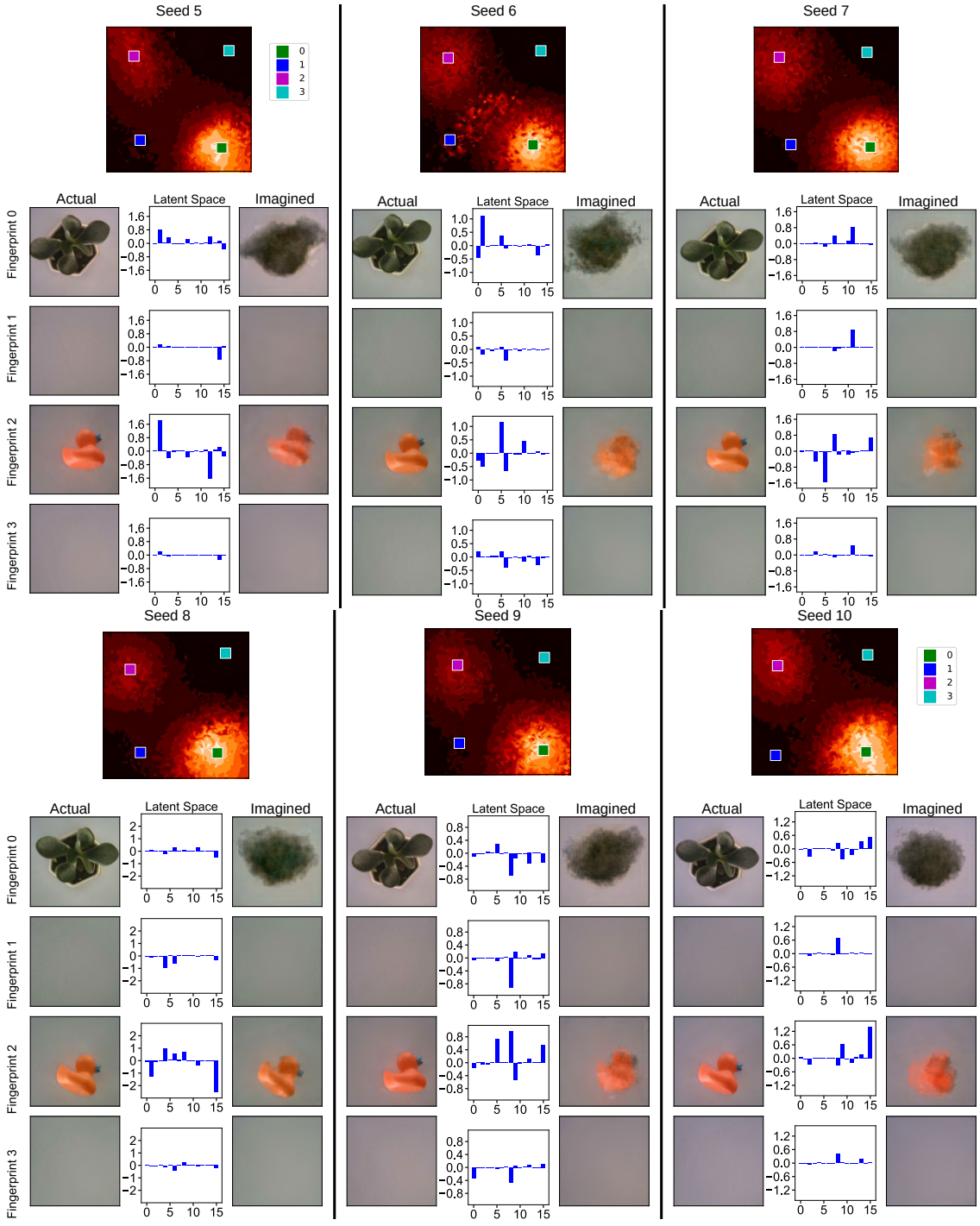

Figure C–2 Fingerprints for Seeds 5-10. See previous figure for seeds 2-4 and caption.

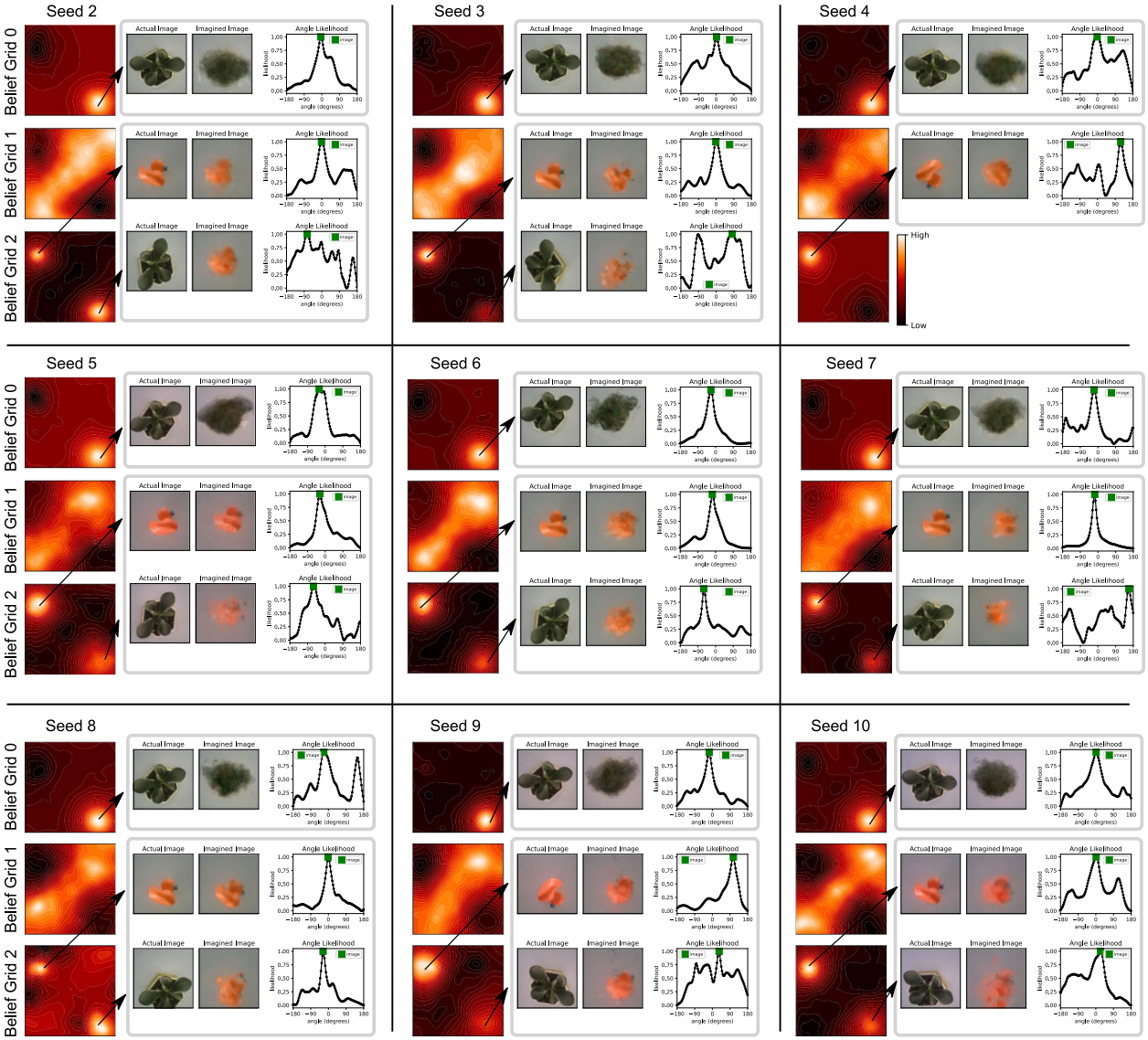

Figure C–3 Actively Learned Belief in Training Environment for Seeds 2-10. Each panel displays identification information for actively learned models with different seeds. The first column shows belief grids for each object with high-likelihood regions in white and low-likelihood regions in black. See Fig. C–1 and C–2 for learned object locations. For Grids 0 and 2, pop-outs are provided for each high-likelihood region with additional information at that x-y location—actual image, imagined image, and normalized angle likelihoods.

C.3 Ablation

This section includes the ablation figures for the seeds and heights excluded from the main text. The ablations in each figure in this section are in the following order:

(1) fixed ramp β and $\gamma = 0$ (solid green bar),

(2) adaptive β and $\gamma = 0$ (solid blue bar),

(3) fixed ramp β and adaptive γ (hatched green bar), and

(4) adaptive β and adaptive γ (hatched blue bar).

Figs. C–5 to C–40 show reconstructions with real and fake inputs by seed. Fake inputs were generated by replacing either the seed state or image with all 0's, samples drawn uniformly from $[0, 1]$, or all 1's (see Fig C–4 for sample ablation images).

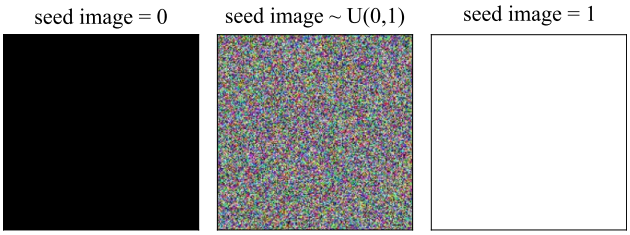

Figure C–4 Ablation images.

Seed 0, z = 0.25

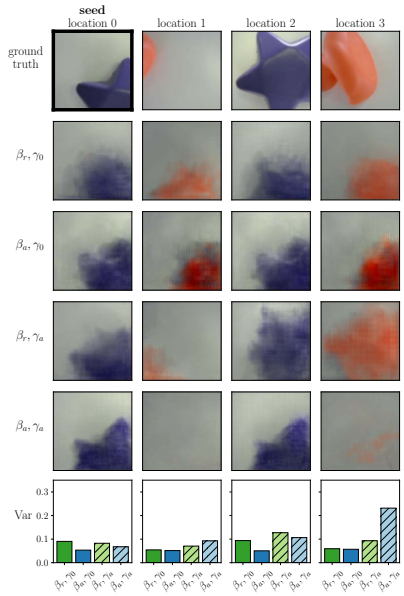

Figure C–5 CVAE reconstruction ablation results (seed 0, $z = 0.25$). The top row shows the ground truth measurements for each subsequent row. The seed image and state is indicated with a thick black box. The next four rows show reconstructions for each loss hyperparameter ablation. The final row shows the predicted variance for each image.

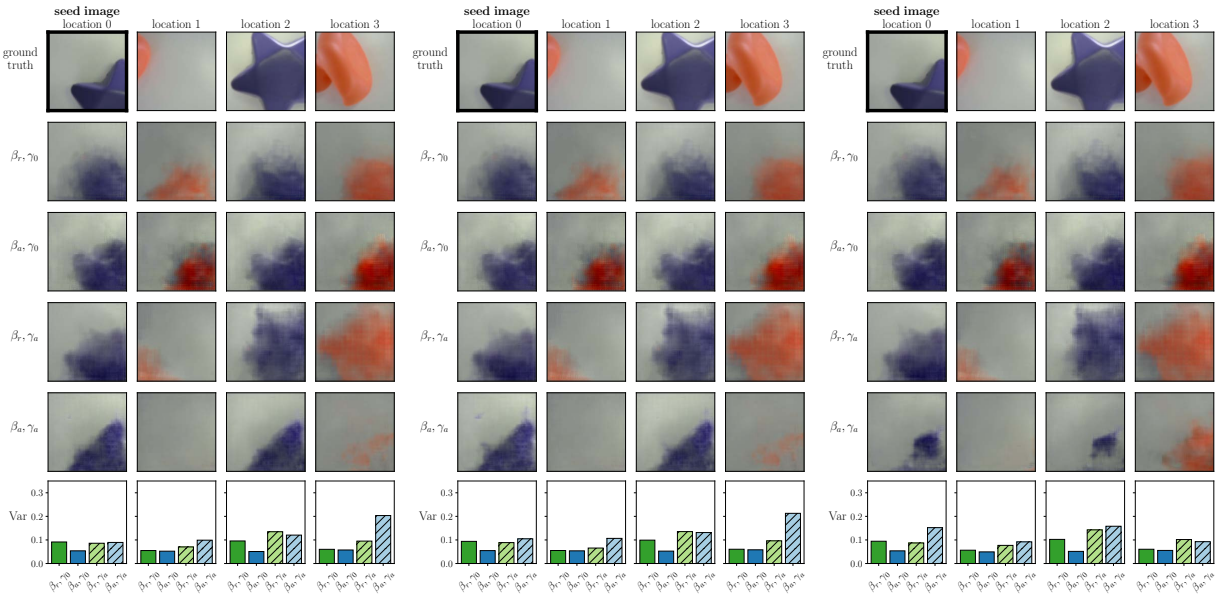

Figure C–6 CVAE state ablation results (seed 0, $z = 0.25$). See Fig. C–5 for description. From left to right, the columns show ablations with the seed x replaced with all 0's, uniform samples from [0,1], and all 1's.

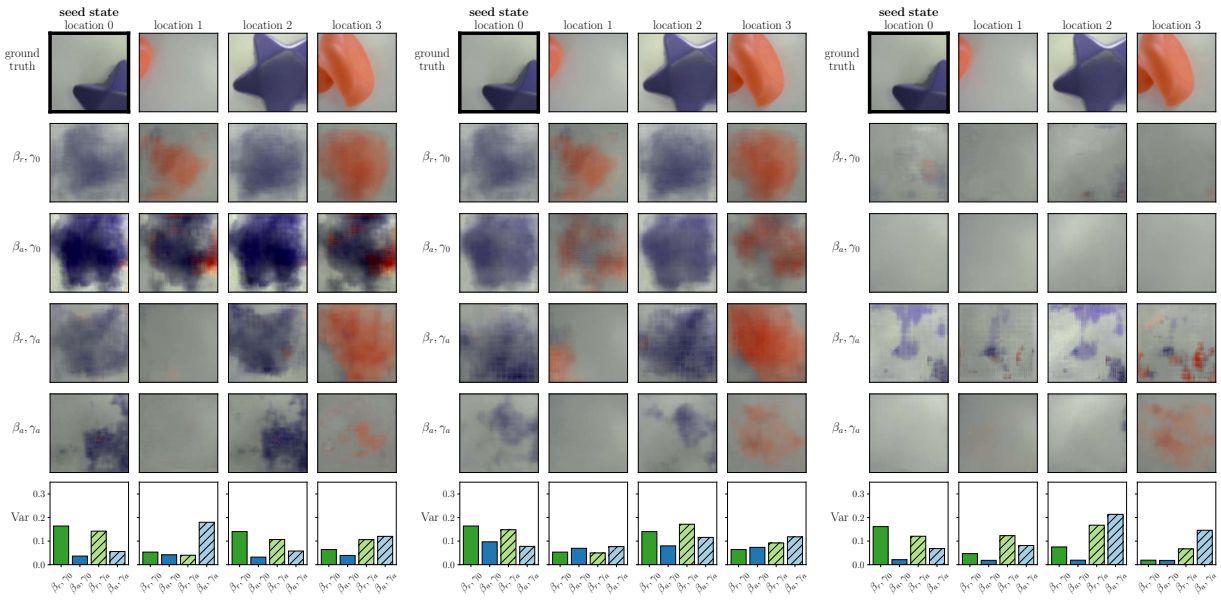

Figure C–7 CVAE image ablation results (seed 0, $z = 0.25$). See Fig. C–5 for description. From left to right, the columns show ablations with the seed image replaced with all 0's, uniform samples from [0,1], and all 1's.

Seed 1, z = 0.25

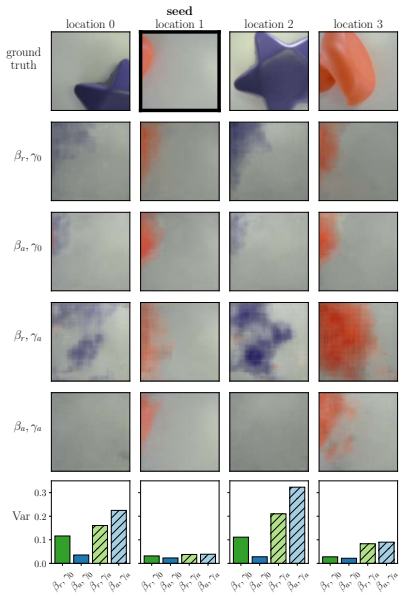

Figure C–8 CVAE reconstruction ablation results (seed 1, $z = 0.25$). See Fig. C–5 for description.

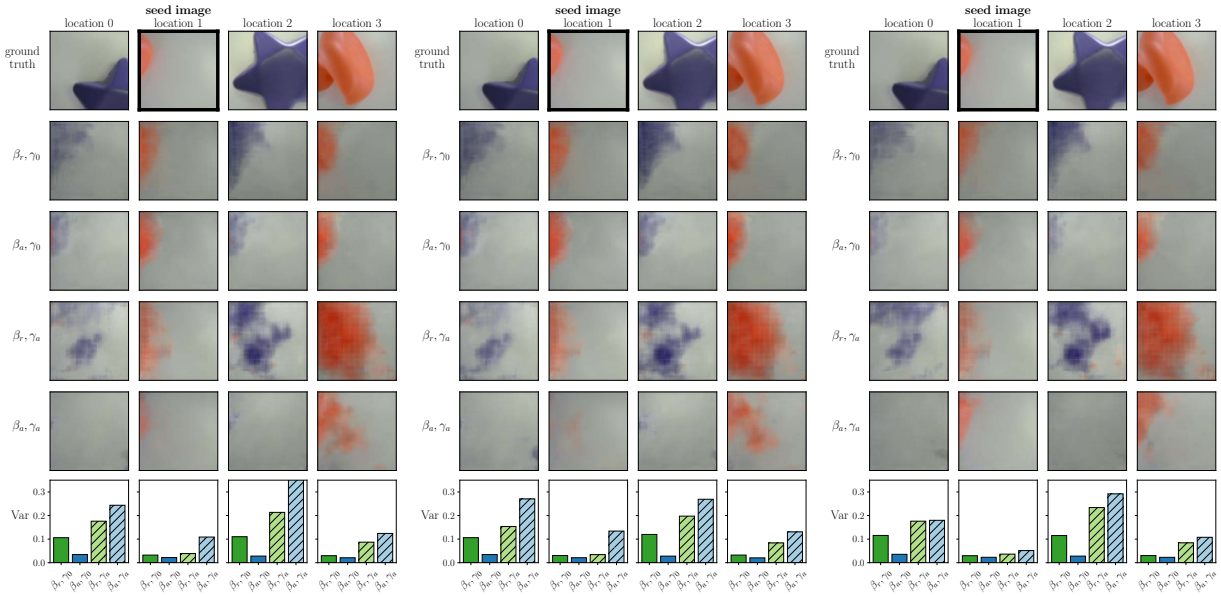

Figure C–9 CVAE state ablation results (seed 1, $z = 0.25$). See Fig. C–6 for description.

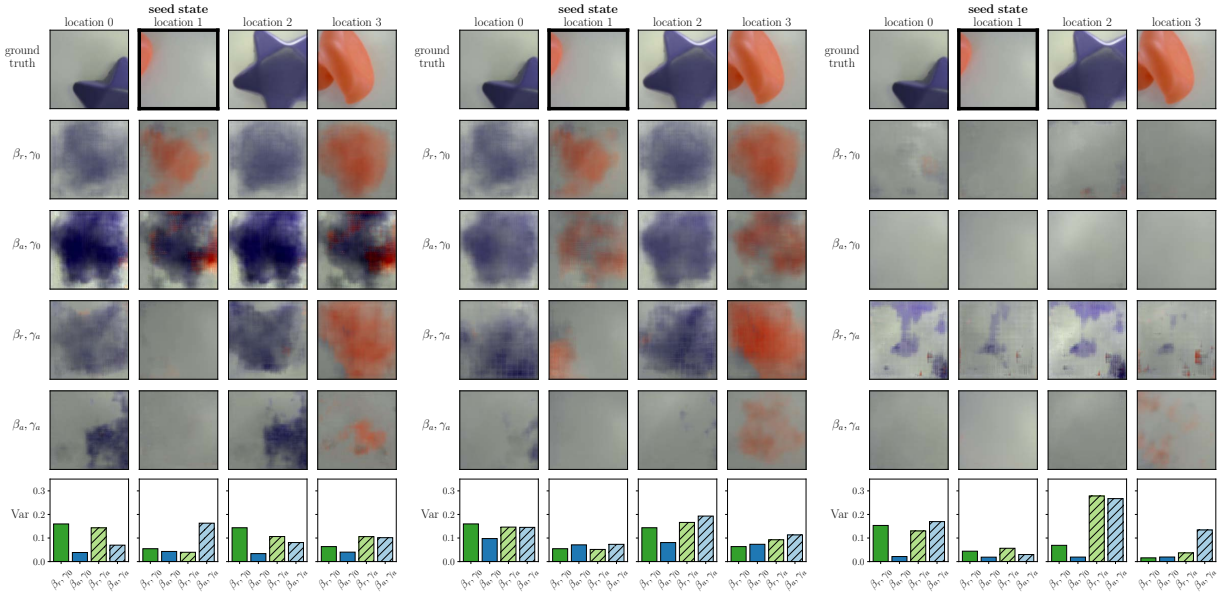

Figure C–10 CVAE image ablation results (seed 1, $z = 0.25$). See Fig. C–7 for description.

Seed 2, z = 0.25

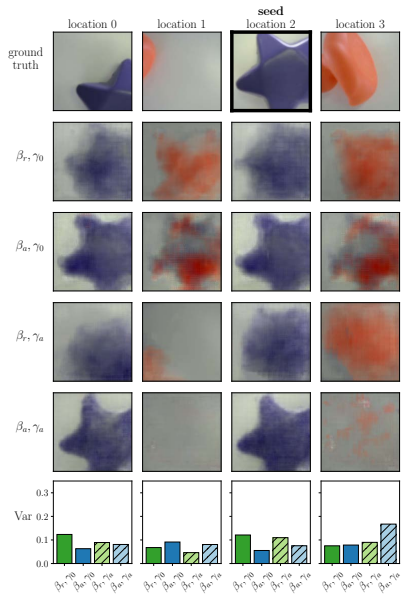

Figure C–11 CVAE reconstruction ablation results (seed 2, $z = 0.25$). See Fig. C–5 for description.

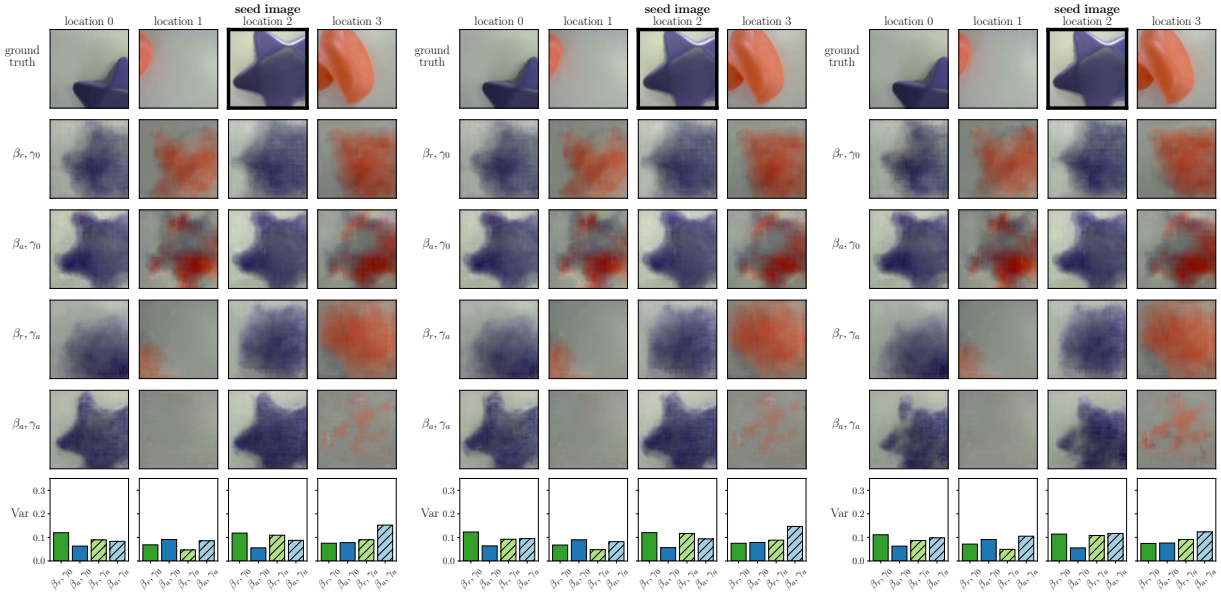

Figure C–12 CVAE state ablation results (seed 2, $z = 0.25$). See Fig. C–6 for description.

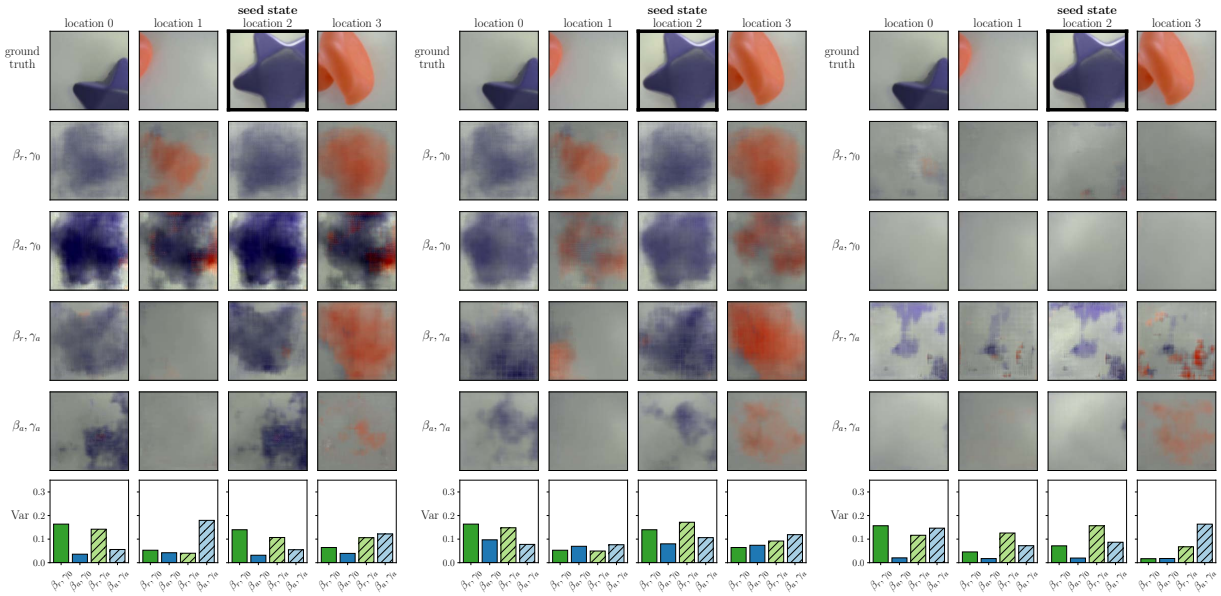

Figure C–13 CVAE image ablation results (seed 2, $z = 0.25$). See Fig. C–7 for description.

Seed 3, z = 0.25

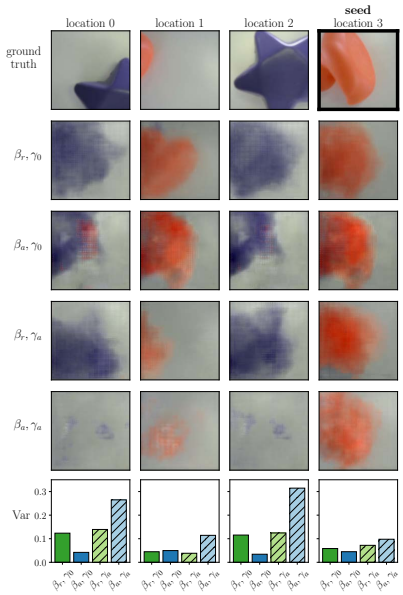

Figure C–14 CVAE reconstruction ablation results (seed 3, $z = 0.25$). See Fig. C–5 for description.

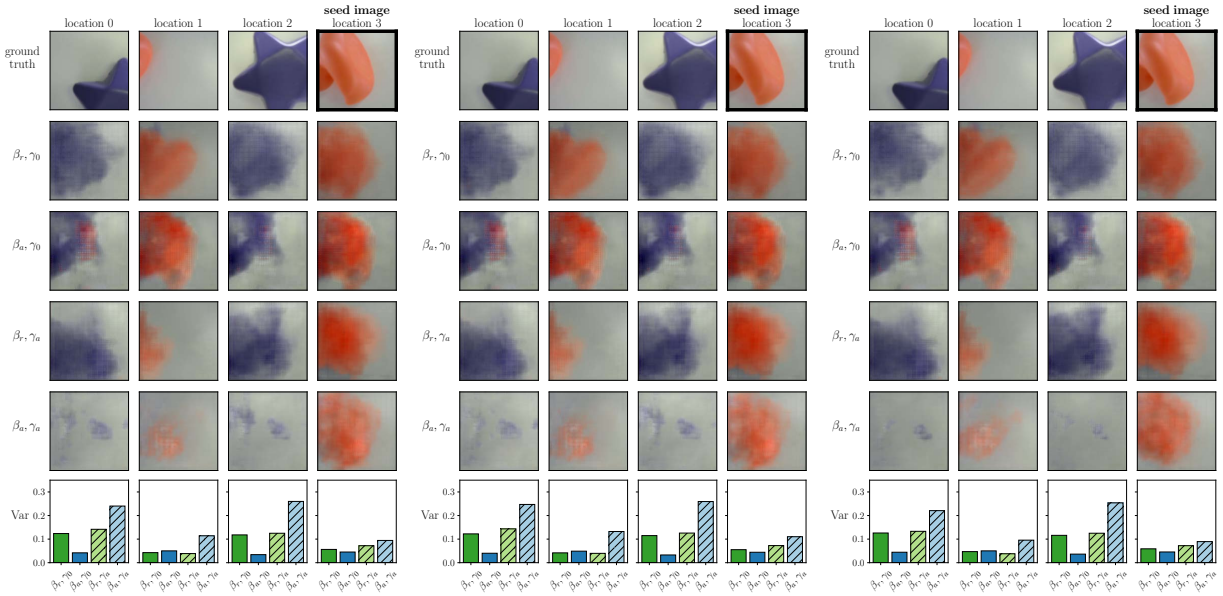

Figure C–15 CVAE state ablation results (seed 3, $z = 0.25$). See Fig. C–6 for description.

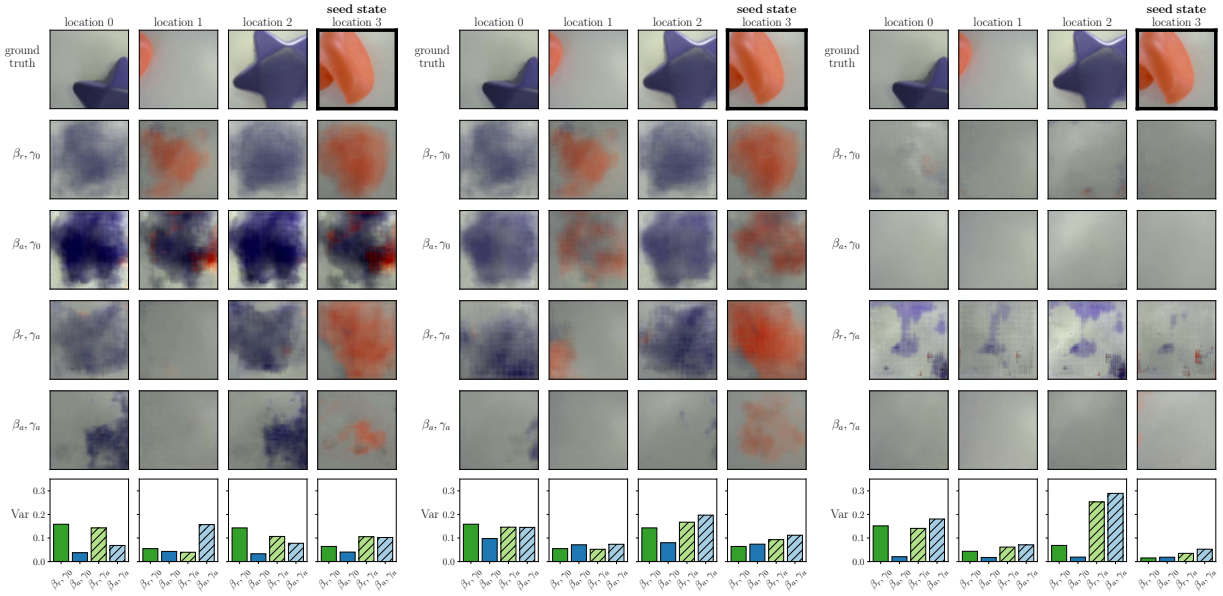

Figure C–16 CVAE image ablation results (seed 3, $z = 0.25$). See Fig. C–7 for description.

Seed 0, z = 0.35

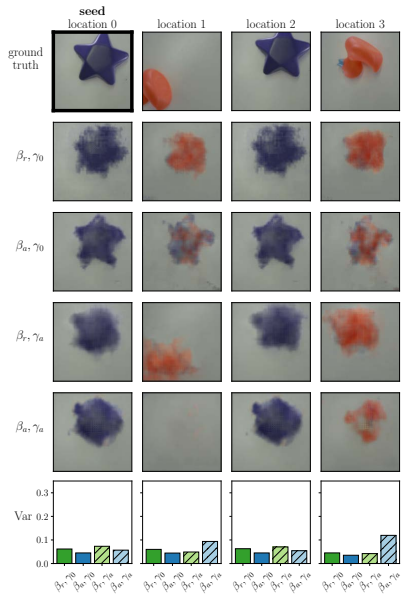

Figure C–17 CVAE reconstruction ablation results (seed 0, $z = 0.35$). See Fig. C–5 for description.

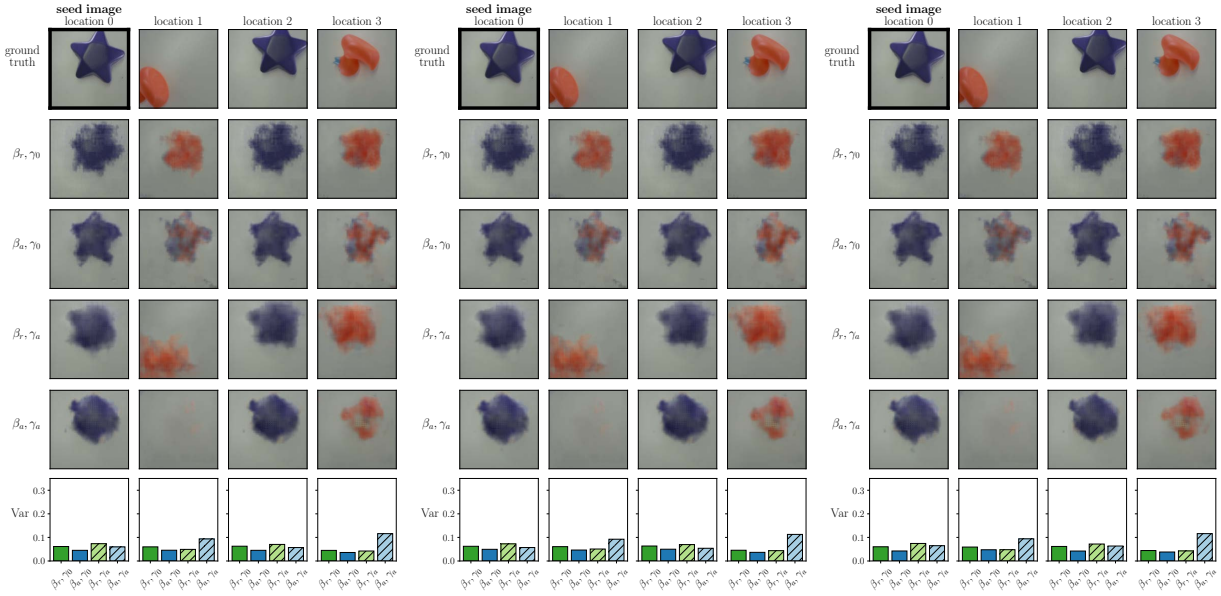

Figure C–18 CVAE state ablation results (seed 0, $z = 0.35$). See Fig. C–6 for description. From left to right, the columns show ablations with the seed x replaced with all 0's, uniform samples from [0,1], and all 1's.

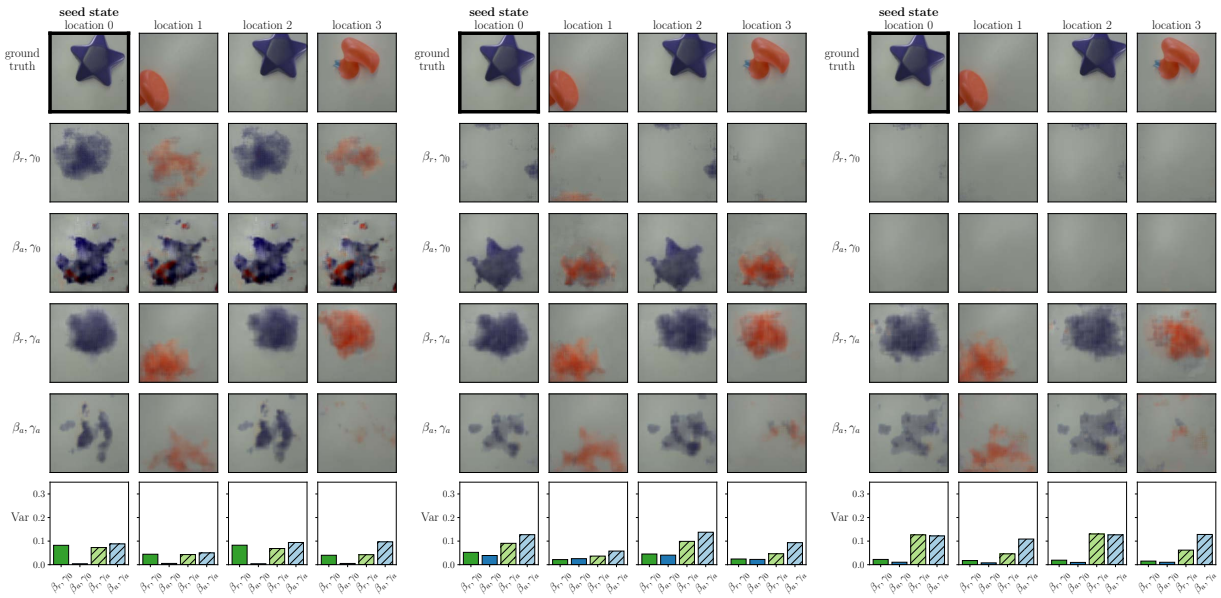

Figure C–19 CVAE image ablation results (seed 0, $z = 0.35$). See Fig. C–7 for description.

Seed 1, z = 0.35

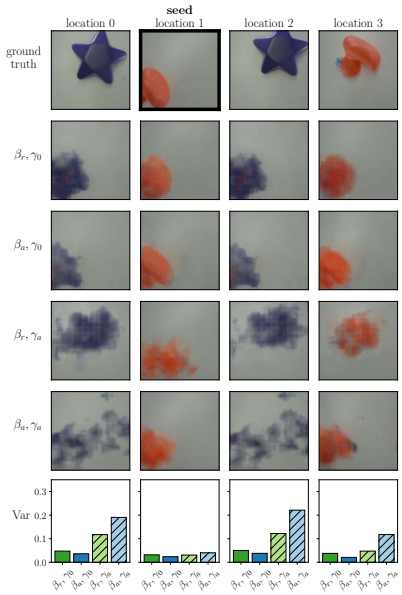

Figure C–20 CVAE reconstruction ablation results (seed 1, $z = 0.35$). See Fig. C–5 for description.

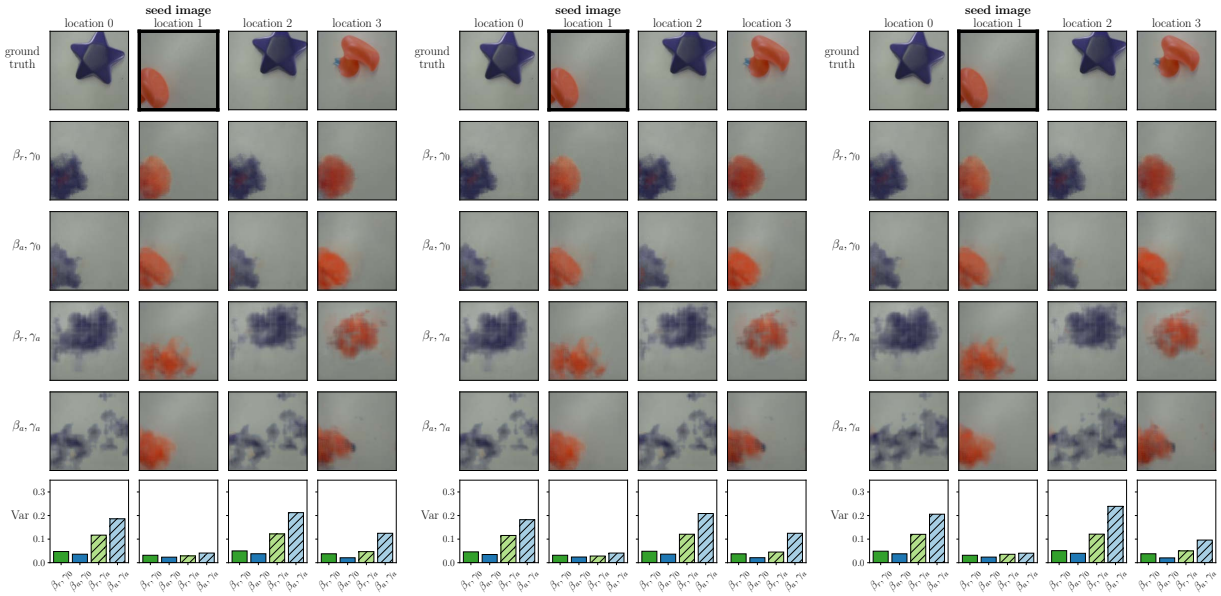

Figure C–21 CVAE state ablation results (seed 1, $z = 0.35$). See Fig. C–6 for description.

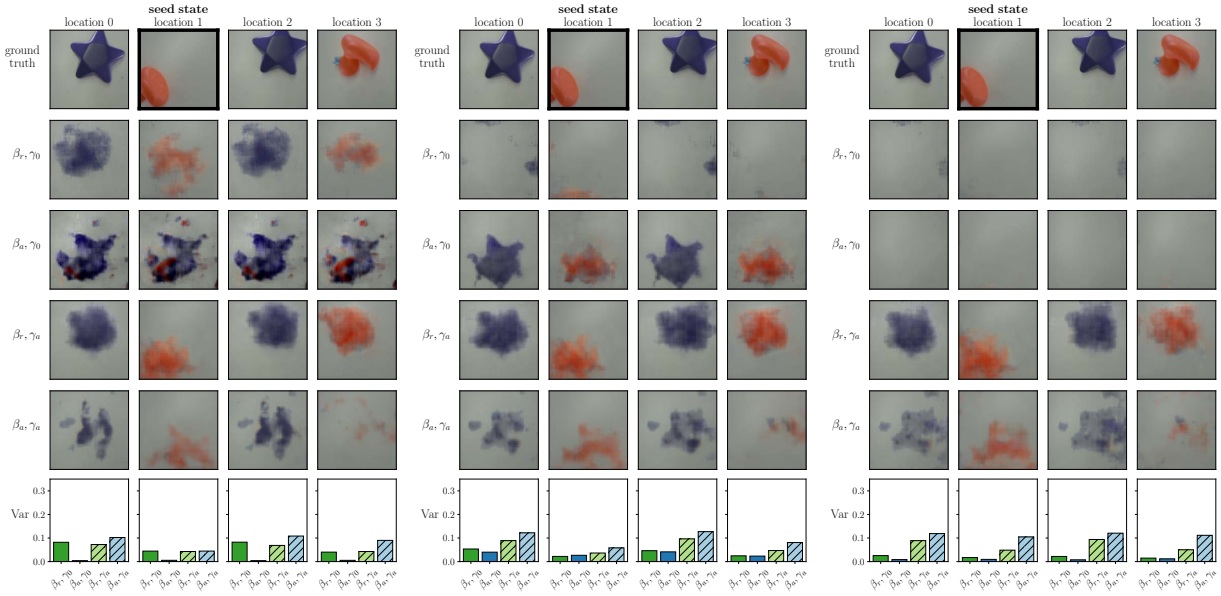

Figure C–22 CVAE image ablation results (seed 1, $z = 0.35$). See Fig. C–7 for description.

Seed 2, z = 0.35

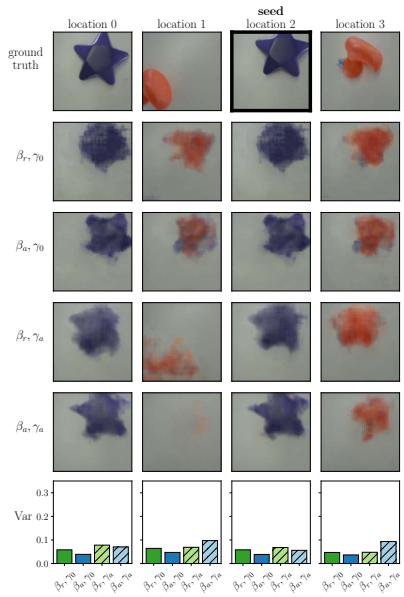

Figure C–23 CVAE reconstruction ablation results (seed 2, $z = 0.35$). See Fig. C–5 for description.

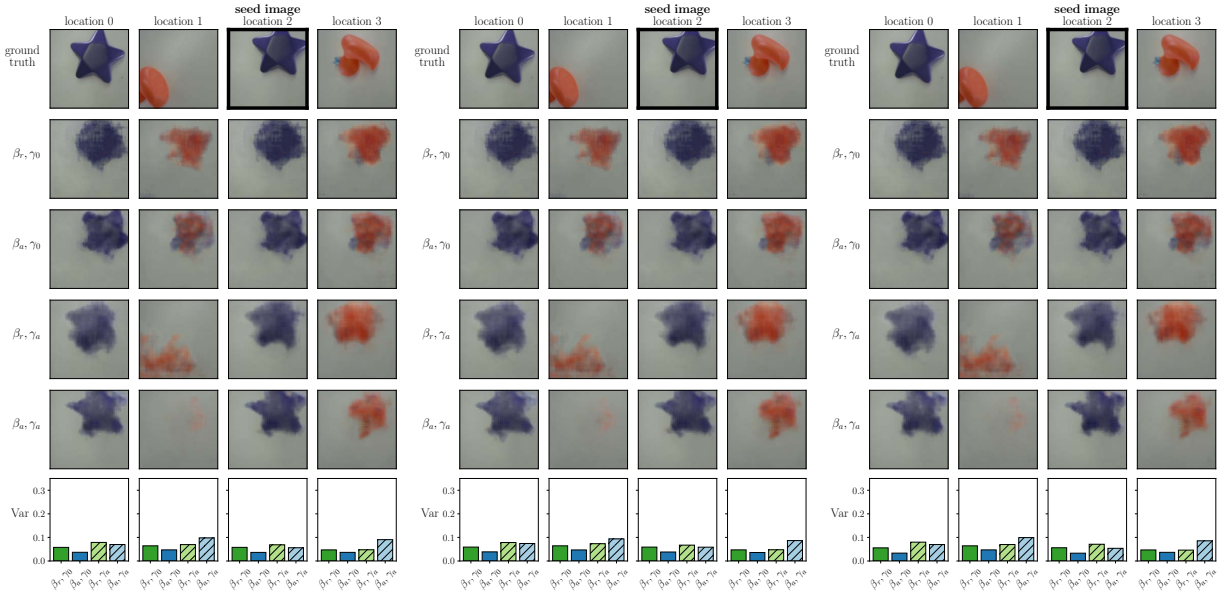

Figure C–24 CVAE state ablation results (seed 2, $z = 0.35$). See Fig. C–6 for description.

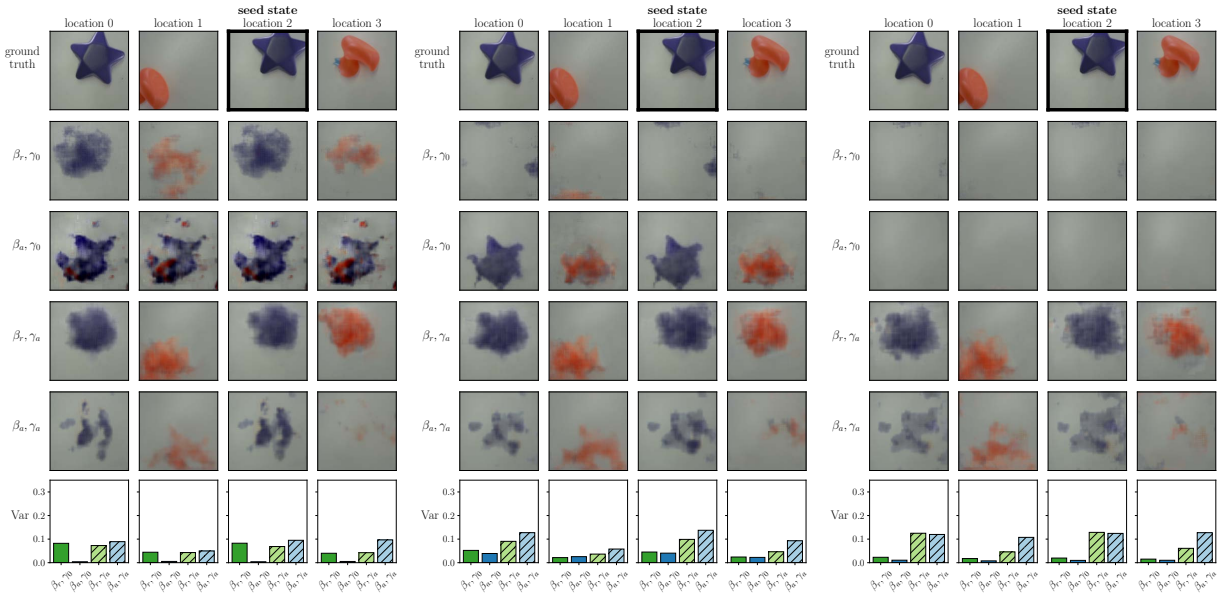

Figure C–25 CVAE image ablation results (seed 2, $z = 0.35$). See Fig. C–7 for description.

Seed 3, z = 0.35

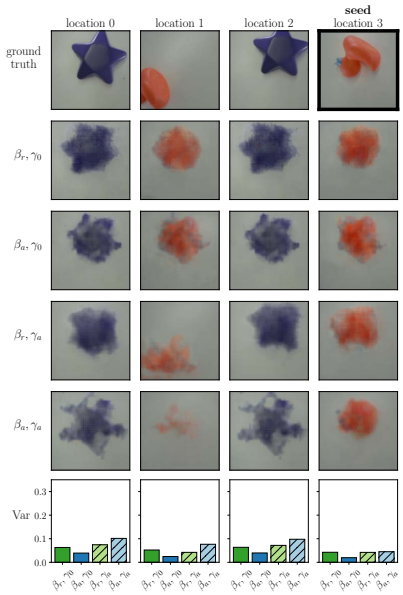

Figure C–26 CVAE reconstruction ablation results (seed 3, $z = 0.35$). See Fig. C–5 for description.

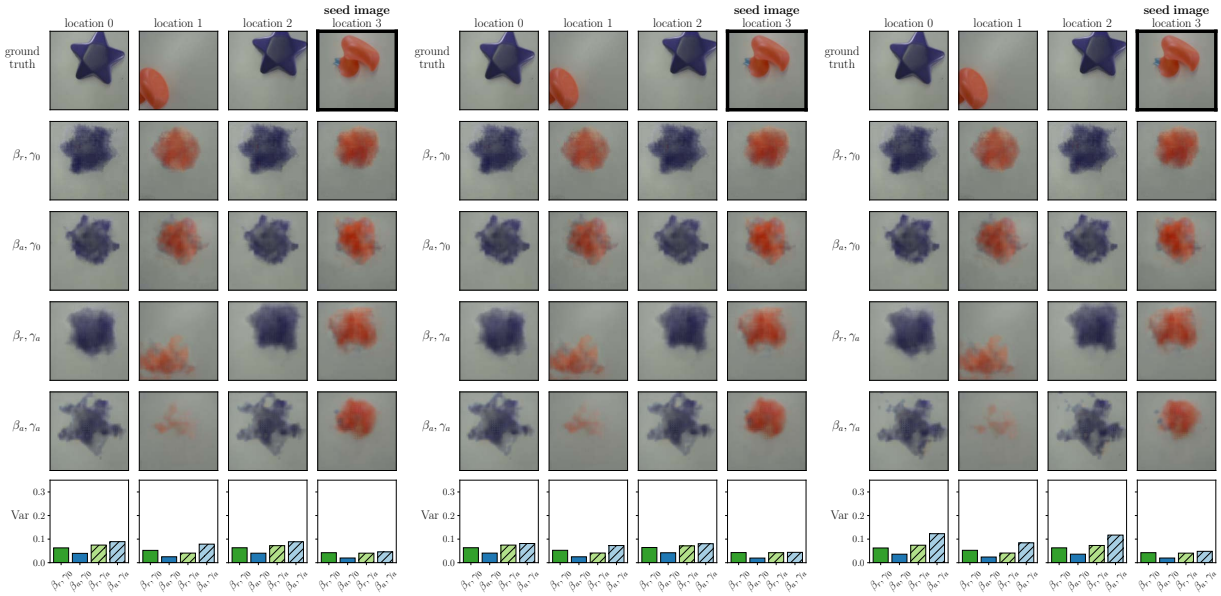

Figure C–27 CVAE state ablation results (seed 3, $z = 0.35$). See Fig. C–6 for description.

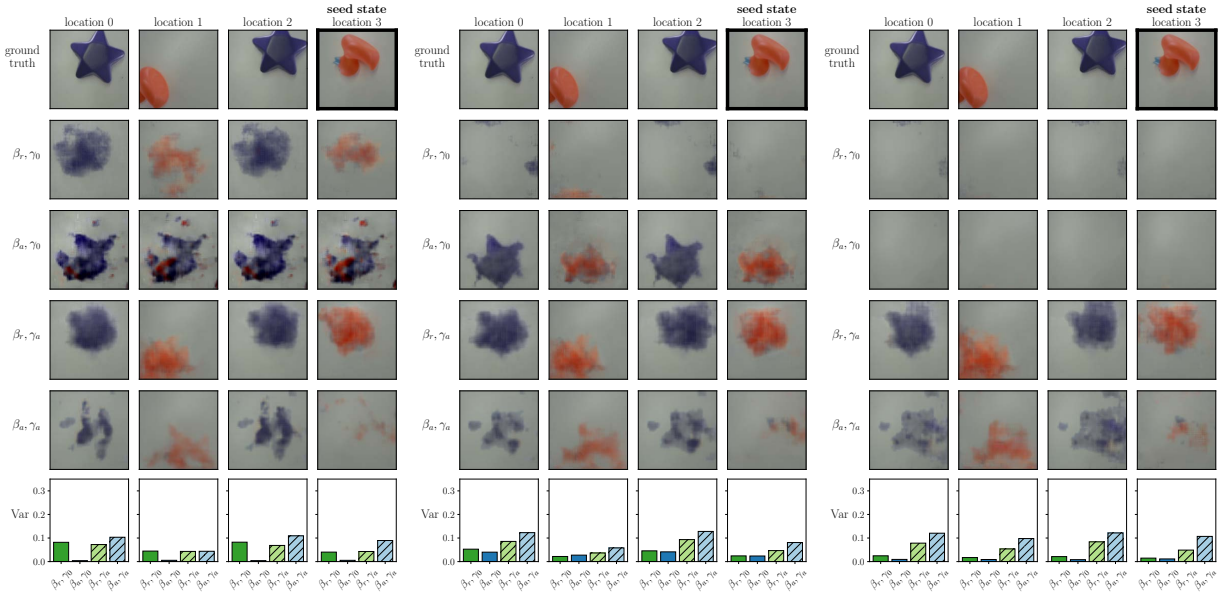

Figure C–28 CVAE image ablation results (seed 3, $z = 0.25$). See Fig. C–7 for description.

Seed 0, z = 0.45

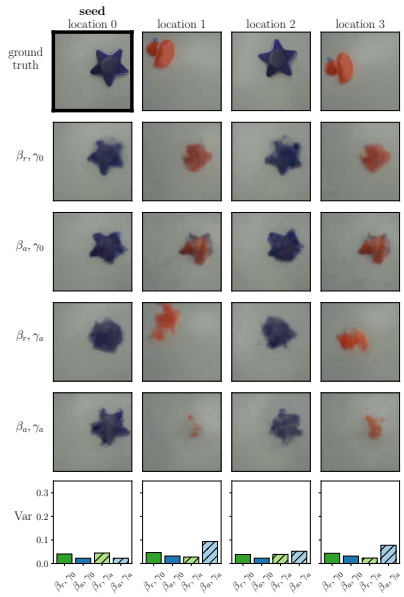

Figure C–29 CVAE reconstruction ablation results (seed 0, $z = 0.45$). See Fig. C–5 for description.

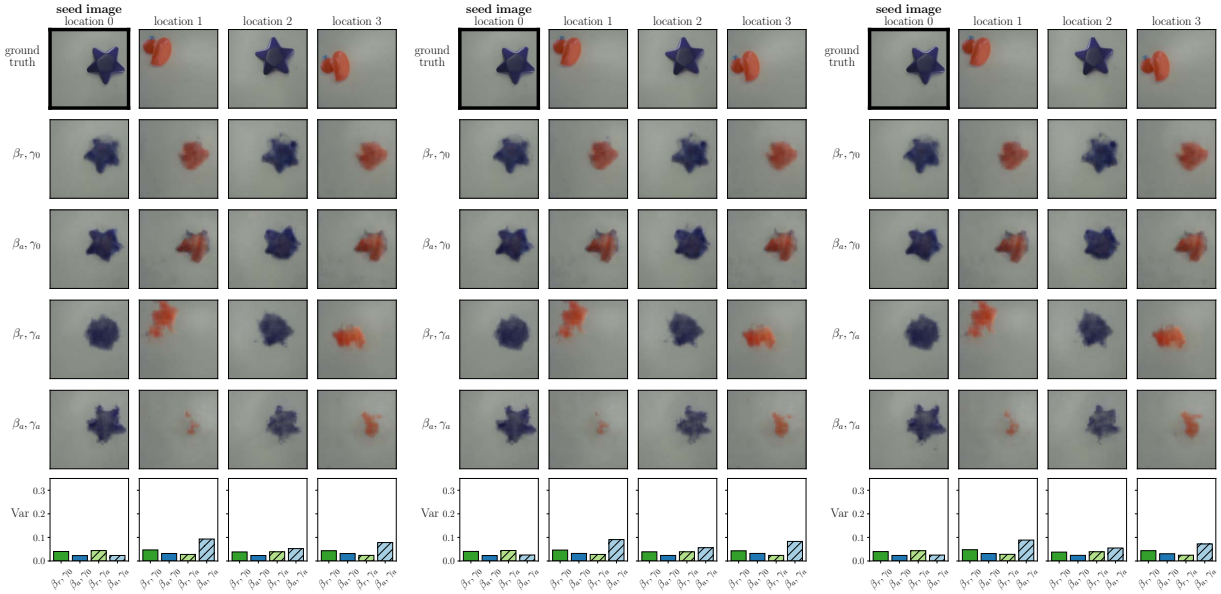

Figure C–30 CVAE state ablation results (seed 0, $z = 0.45$). See Fig. C–6 for description. From left to right, the columns show ablations with the seed x replaced with all 0's, uniform samples from [0,1], and all 1's.

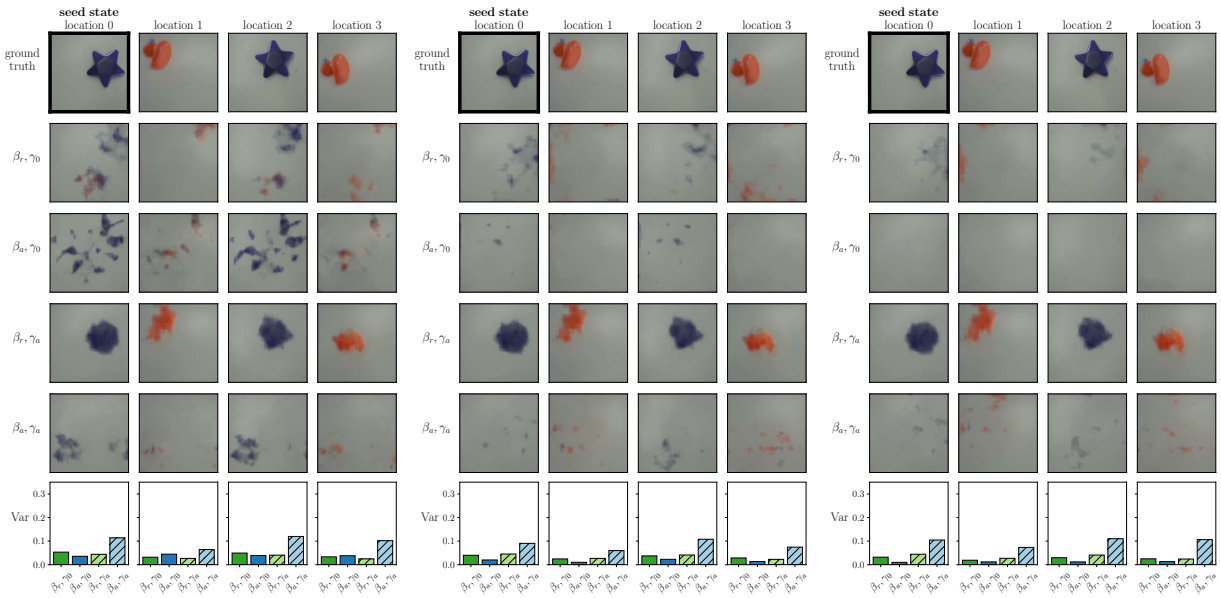

Figure C–31 CVAE image ablation results (seed 0, $z = 0.45$). See Fig. C–7 for description.

Seed 1, z = 0.45

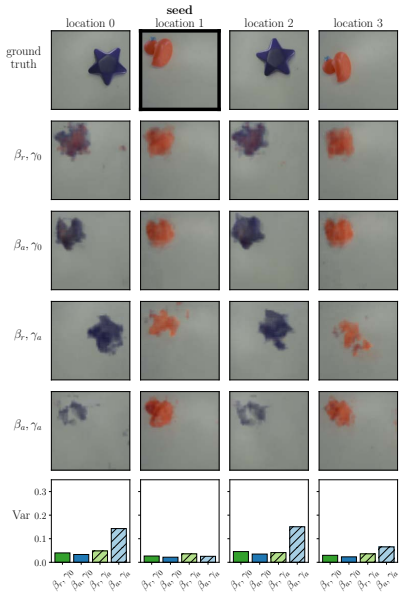

Figure C–32 CVAE reconstruction ablation results (seed 1, $z = 0.45$). See Fig. C–5 for description.

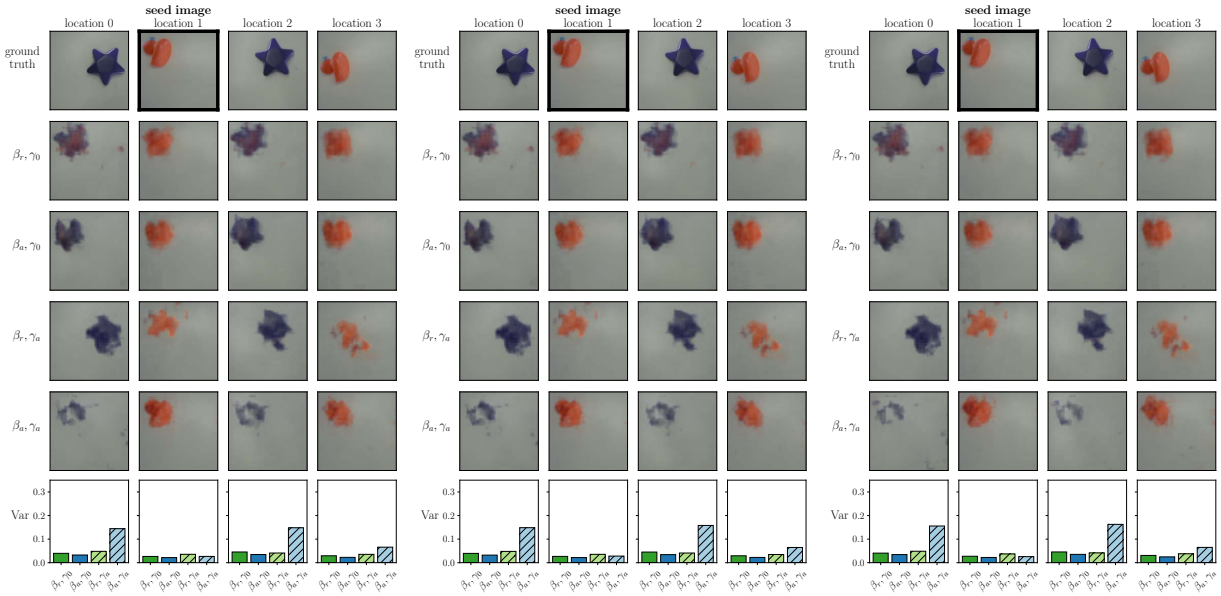

Figure C–33 CVAE state ablation results (seed 1, $z = 0.45$). See Fig. C–6 for description.

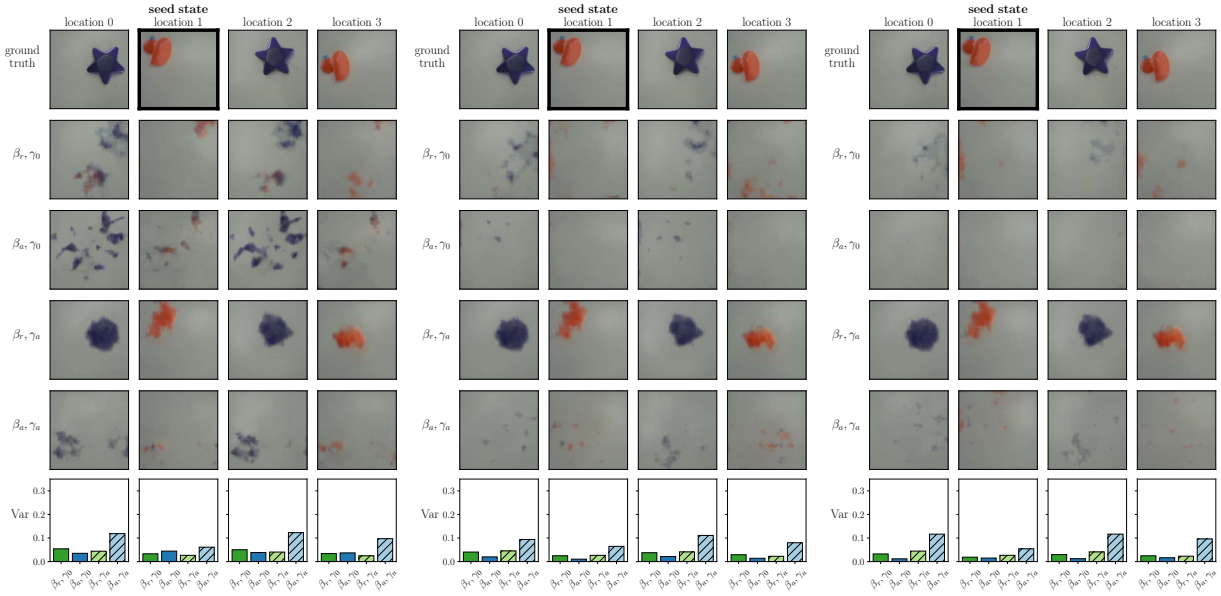

Figure C–34 CVAE image ablation results (seed 1, $z = 0.45$). See Fig. C–7 for description.

Seed 2, z = 0.45

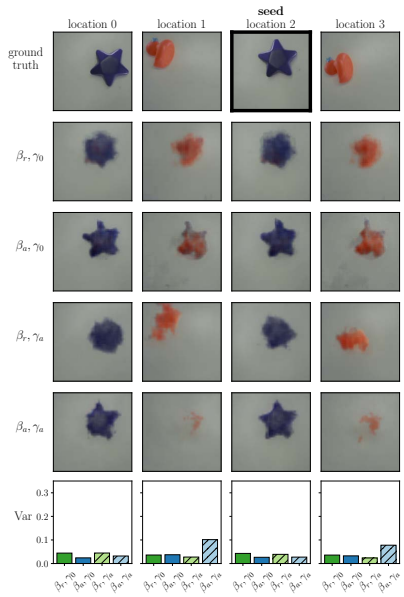

Figure C–35 CVAE reconstruction ablation results (seed 2, $z = 0.45$). See Fig. C–5 for description.

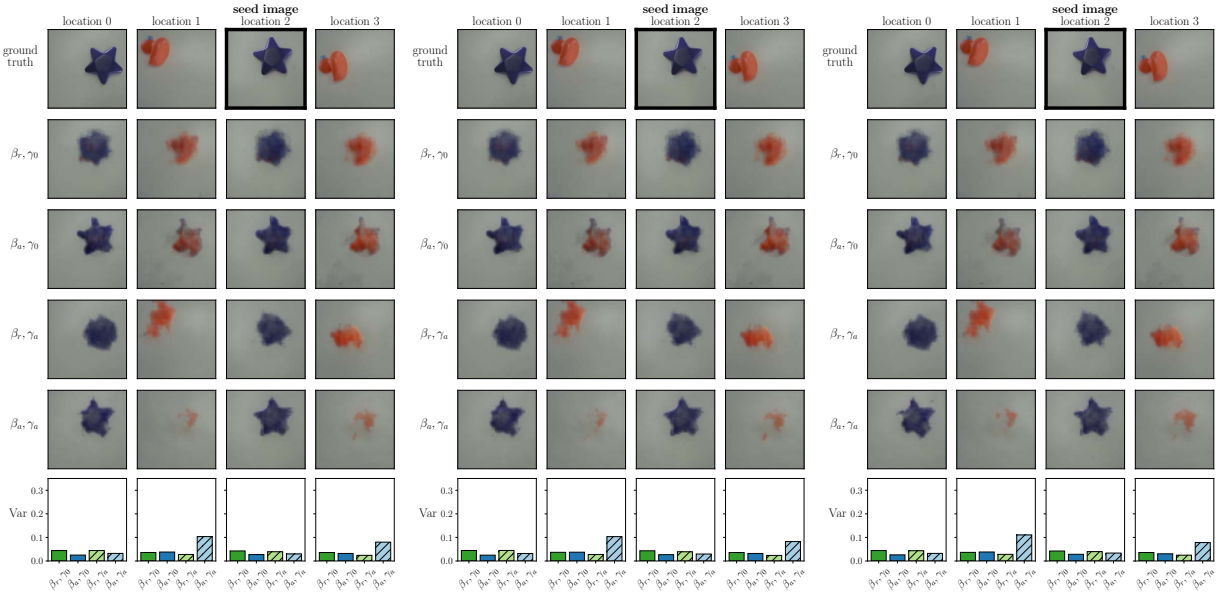

Figure C–36 CVAE state ablation results (seed 2, $z = 0.45$). See Fig. C–6 for description.

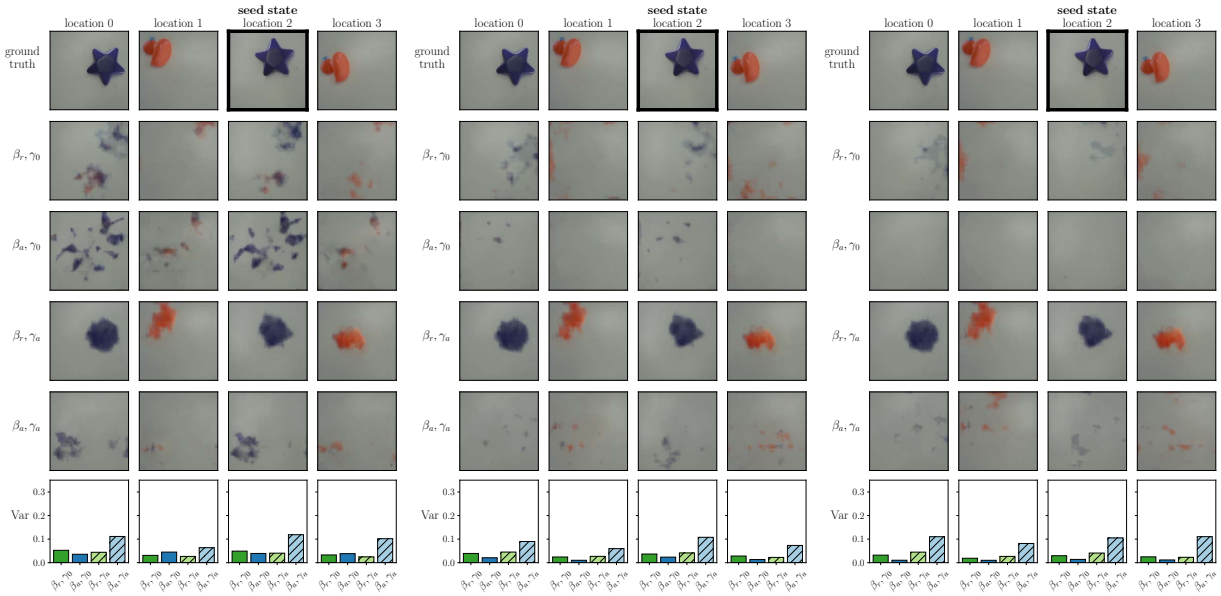

Figure C–37 CVAE image ablation results (seed 2, $z = 0.45$). See Fig. C–7 for description.

Seed 3, z = 0.45

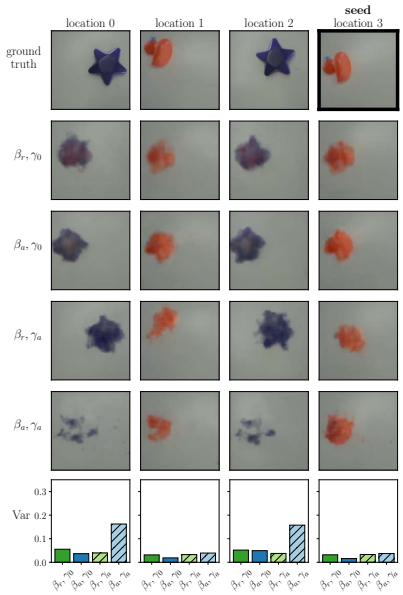

Figure C–38 CVAE reconstruction ablation results (seed 3, $z = 0.45$). See Fig. C–5 for description.

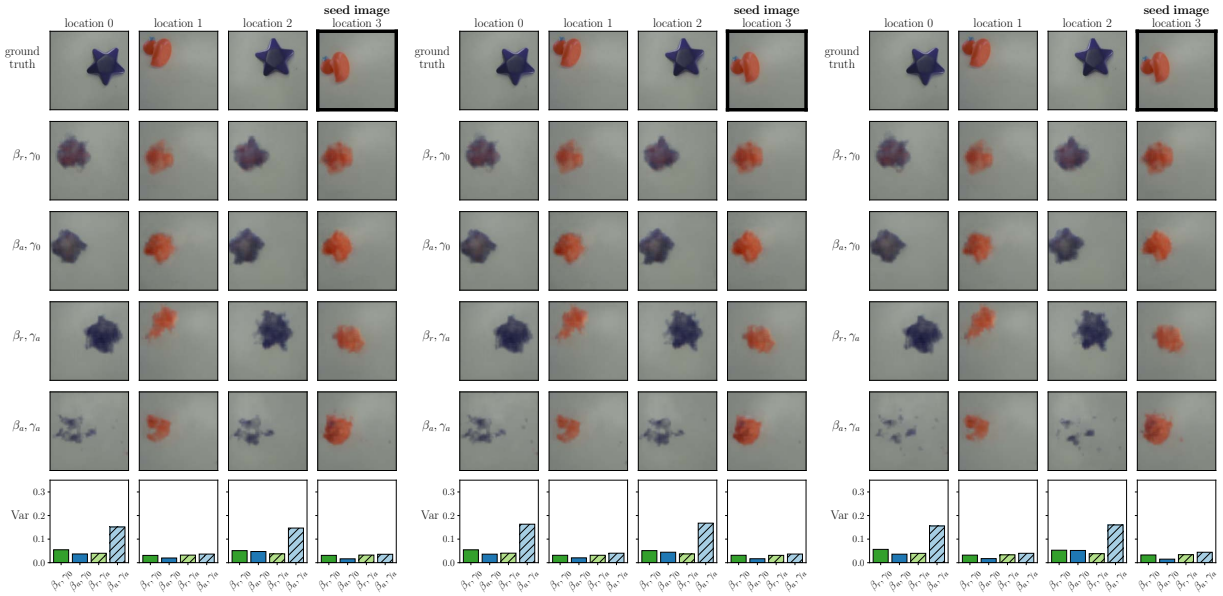

Figure C–39 CVAE state ablation results (seed 3, $z = 0.45$). See Fig. C–6 for description.

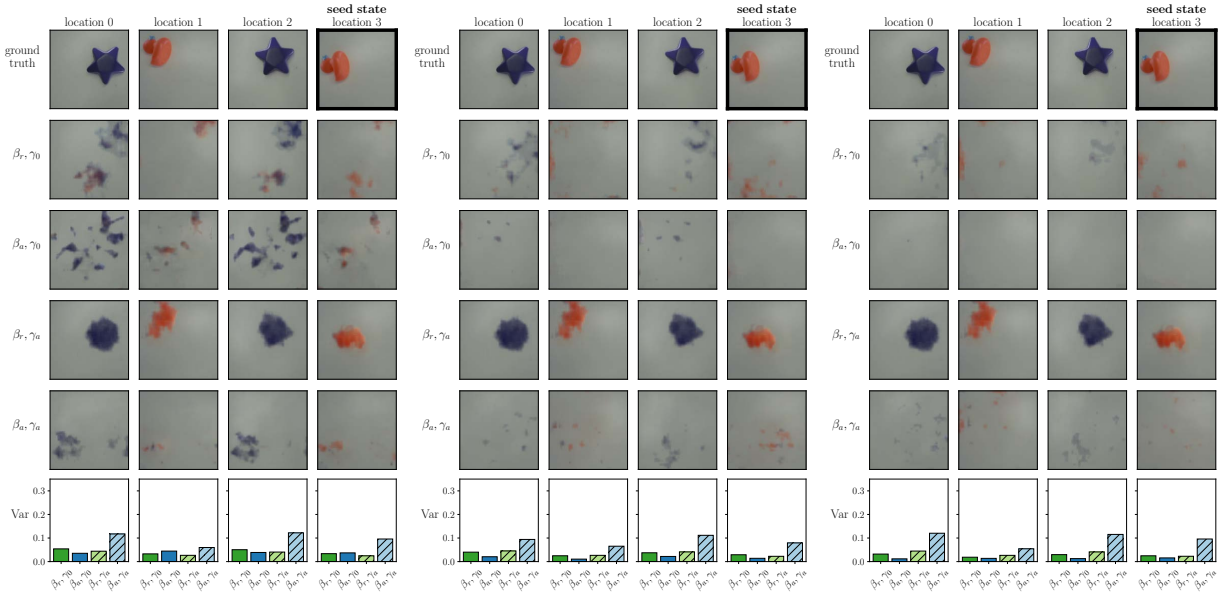

Figure C–40 CVAE image ablation results (seed 3, $z = 0.25$). See Fig. C–7 for description.

C.4 GelSight Experiments

Table C–5 summarizes the workspace parameters used for the environments in Fig. 6–20. We used adaptive β and γ for these experiments, and we reduced the trainer update throttle to 2 updates per control step instead of 3. All other parameters are the same as the previously listed values. To prevent excessive forces, we cap velocities when a maximum force of 5 is measured in the desired force direction. If the robot gets stuck due to the velocity clipping, we override commands the robot to move with velocity opposite the current force vector. Another issue with end effector control is that the robot can get stuck at singularities or joint limits that the robot is unaware of. In such instances, we manually override the controls to get the robot unstuck and continue learning.

Parameter	**Ball + Duck**	**Dog Toy**	**Skull**
x	$[0.325\text{ m}, 0.625\text{ m}]$	$[0.325\text{ m}, 0.625\text{ m}]$	$[0.325\text{ m}, 0.625\text{ m}]$
y	$[-0.11\text{ m}, 0.11\text{ m}]$	$[-0.11\text{ m}, 0.11\text{ m}]$	$[-0.11\text{ m}, 0.11\text{ m}]$
z	$[0.15\text{ m}, 0.25\text{ m}]$	$[0.12\text{ m}, 0.22\text{ m}]$	$[0.15\text{ m}, 0.45\text{ m}]$
pitch	$[-0.25\text{ rad}, 0.25\text{ rad}]$	$[-0.25\text{ rad}, 0.25\text{ rad}]$	$[-0.25\text{ rad}, 0.25\text{ rad}]$

Table C–5 Franka Workspace Parameters for GelSight Experiments by Environment