ABSTRACT

Title of Dissertation: TOWARDS ROBUST AND ADAPTABLE REAL-WORLD REINFORCEMENT LEARNING

Yanchao Sun
Doctor of Philosophy, 2023

Dissertation Directed by: Doctor Furong Huang
Department of Computer Science

The past decade has witnessed a rapid development of reinforcement learning (RL) techniques. However, there is still a gap between employing RL in simulators and applying RL models to challenging and diverse real-world systems. On the one hand, existing RL approaches have been shown to be fragile under perturbations in the environment, making it risky to deploy RL models in real-world applications where unexpected noise and interference exist. On the other hand, most RL methods focus on learning a policy in a fixed environment, and need to re-train a policy if the environment gets changed. For real-world environments whose agent specifications and dynamics can be ever-changing, these methods become less practical as they require a large amount of data and computations to adapt to a changed environment.

We focus on the above two challenges and introduce multiple solutions to improve the robustness and adaptability of RL methods. For robustness, we propose a series of approaches that define, explore, and mitigate the vulnerability of RL agents from different perspectives and achieve state-of-the-art performance on robustifying RL policies. For adaptability, we present

transfer learning and pretraining frameworks to address challenging multi-task learning problems that are important yet rarely studied, contributing to the application of RL techniques to more real-life scenarios.

TOWARDS ROBUST AND ADAPTABLE REAL-WORLD REINFORCEMENT LEARNING

by

Yanchao Sun

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Dr. Furong Huang, Chair/Advisor
Prof. Min Wu, Dean's Representative
Prof. Dinesh Manocha
Prof. Hal Daumé III
Prof. Tom Goldstein
Dr. Kaiqing Zhang

Acknowledgments

I owe my gratitude to all the people who have supported me throughout my Ph.D. journey. This dissertation would not have been possible without their help.

First and foremost, I would like to express my deepest appreciation to my advisor, Dr. Furong Huang, for her unwavering guidance, support, and encouragement throughout my research. Dr. Huang's expertise and insights have been invaluable in shaping my ideas and approach to my research. Beyond being my advisor, Dr. Huang has been a mentor, role model, and friend. Her guidance has helped me grow both personally and professionally.

I would like to thank the members of my Ph.D. committee, Prof. Min Wu, Prof. Dinesh Manocha, Prof. Hal Daumé III, Prof. Tom Goldstein, and Dr. Kaiqing Zhang. Their valuable time, feedback, and suggestions have helped me a lot to improve my research, dissertation, and presentation. I would also like to thank Dr. Soheil Feizi for serving on the committee of my preliminary oral exam and providing insightful advice on my research.

I would like to express my appreciation to my internship mentors, Dr. Shuang Ma, Dr. Sumitra Ganesh, and Dr. Andrew Cohen, who provided me with generous guidance and support during my internships at Microsoft Research, JPMorgan AI Research, and Unity Technologies. I am grateful to all my collaborators and managers from these programs for their valuable contributions and insights.

I enjoyed the collaboration and the interaction with other talented students from our re-

search lab. I am deeply grateful for all their contributions and inputs. I am also honored to collaborate with Dr. Tianyi Zhang, Dr. Lichao Sun and their students on several projects. I would also like to thank Dr. Fakai Wang and Prof. Min Wu for providing the LaTex style files and formatting tips for writing this thesis.

I also owe my gratitude to Prof. Ning Yang and Prof. Philip S. Yu, who guided me in my first research project before applying for the Ph.D. program. Without their help and encouragement, I might not have discovered my passion for artificial intelligence and chosen to pursue a Ph.D.

Finally, I would like to express my heartfelt gratitude to my family and friends, who have supported me throughout my Ph.D. journey with their love, encouragement, and patience. I am particularly grateful to my boyfriend, Yuxiang, for his unwavering support and companionship during this journey. Words cannot be enough to express my gratitude to my parents and my brother in China. The pandemic made it hard for us to reunite, but their understanding and love are the sources of my motivation and courage. Lastly, thank you to my beloved cat, Aspen, for bringing joy (and occasional distractions) to my life and work.

I apologize if I have inadvertently left out anyone who has contributed to my journey. Your support has been invaluable and has made a significant impact on my life and work.

Table of Contents

Acknowledgements ii

Table of Contents iv

List of Tables viii

List of Figures ix

List of Abbreviations xii

Chapter 1: Introduction 1
- 1.1 Background and Motivation . . . 1
- 1.2 Common Preliminaries and Notations . . . 3
- 1.3 Organization and Overview . . . 4

Chapter 2: Measuring the Robustness of RL Policies 6
- 2.1 Introduction . . . 6
- 2.2 Related Work . . . 9
- 2.3 Background and Problem Setup . . . 10
- 2.4 Proposed Approach . . . 12
  - 2.4.1 Understanding Optimal Adversary via Policy Perturbations . . . 12
  - 2.4.2 PA-AD: Optimal and Efficient Evasion Attack . . . 16
- 2.5 Empirical Results . . . 21
- 2.6 Conclusion . . . 25
- 2.7 Supplemental Materials: Proofs and Additional Details . . . 26
  - 2.7.1 Relationship between Evasion Attacks and Policy Perturbations . . . 26
  - 2.7.2 Extentions and Additional Details of Our Algorithm . . . 28
  - 2.7.3 Variants For Environments with Continuous Action Spaces . . . 31
  - 2.7.4 Topological Properties of the Admissible Adversarial Policy Set . . . 32
  - 2.7.5 Characterize Optimality of Evasion Attacks . . . 44
  - 2.7.6 Existence of An Optimal Policy Adversary . . . 45
  - 2.7.7 Characterizing Optimality of Heuristic Attacks . . . 50
  - 2.7.8 Additional Experiment Details and Results . . . 59

Chapter 3: A Generic Training Framework for Improving the Robustness of RL 74
- 3.1 Introduction . . . 74
- 3.2 Related Work . . . 77

3.3 Background and Problem Setup . . . 79
3.4 Proposed Approach . . . 80
3.4.1 A Closer Look at Robust RL . . . 80
3.4.2 WocaR-RL: Worst-case-aware Robust RL . . . 83
3.5 Empirical Results . . . 91
3.5.1 Experiments and Evaluations . . . 92
3.5.2 Verifying Effectiveness of WocaR-RL . . . 96
3.6 Conclusion . . . 98
3.7 Supplemental Materials: Proofs and Additional Details . . . 99
3.7.1 Additional Definitions and Proofs . . . 99
3.7.2 Algorithm Details . . . 101
3.7.3 Worst-case-aware Robust A2C (WocaR-A2C) . . . 105
3.7.4 Experiment Details and Additional Results . . . 108

Chapter 4: Robustness of RL under Training-Time Poisoning 121
4.1 Introduction . . . 121
4.2 Related Work . . . 124
4.3 Background and Problem Setup . . . 126
4.3.1 Notations and Preliminaries . . . 126
4.3.2 The Procedure of Online Learning and Poisoning . . . 126
4.4 Proposed Approach . . . 128
4.4.1 A Unified Formulation for Poisoning Online Policy-based RL. . . . 128
4.5 VA2C-P: Poison Policy Gradient Learners . . . 131
4.5.1 Decision 1: When to Attack – Vulnerability-Aware . . . 132
4.5.2 Decision 2: How to Attack – Adversarial Critic . . . 134
4.5.3 Poisoning Algorithm VA2C-P . . . 135
4.6 Empirical Results . . . 137
4.7 Conclusion . . . 140
4.8 Supplemental Materials: Proofs and Additional Details . . . 141
4.8.1 A Generic Poisoning Framework for RL . . . 141
4.8.2 Theoretical Analysis . . . 146
4.8.3 Measuring Test-Time Vulnerability of RL Algorithms via Robustness Radius . . . 147
4.8.4 Vulnerability Comparison: Difference Between SL and RL . . . 148
4.8.5 Algorithm Details . . . 151
4.8.6 Additional Implementation Details and Empirical Results . . . 153

Chapter 5: Robust Communication in MARL Systems 157
5.1 Introduction . . . 157
5.2 Related Work . . . 160
5.3 Background and Problem Setup . . . 162
5.3.1 Communicative Muti-agent Reinforcement Learning (CMARL) . . . 162
5.3.2 Communication Attacks in CMARL . . . 164
5.4 Proposed Approach . . . 165
5.4.1 Ablated Message Ensemble (AME) . . . 165

5.4.2 Robustness Certificates of AME . . . 168
5.4.3 Interpretation of Conditions and Hyperparameter . . . 175
5.4.4 Scaling Up: Ensemble with Partial Samples . . . 176
5.5 Empirical Results . . . 178
5.6 Conclusion . . . 183
5.7 Supplemental Materials: Proofs and Additional Details . . . 183
5.7.1 Technical Proofs . . . 183
5.7.2 Proofs in Discrete Action Space . . . 184
5.7.3 Proofs in Continuous Action Space . . . 185
5.7.4 Additional Implementation Details and Empirical Results . . . 187

Chapter 6: Adapting to Novel Environments with Theoretical Guarantees 192
6.1 Introduction . . . 192
6.2 Related Work . . . 195
6.3 Background and Problem Setup . . . 197
6.4 Proposed Approach . . . 197
6.4.1 Learning with Templates . . . 197
6.4.2 O-TempLe: Online Template Learning . . . 202
6.4.3 FM-TempLe: Finite-Model Template Learning . . . 205
6.4.4 Theoretical Analysis . . . 206
6.5 Empirical Results . . . 210
6.5.1 Finite-Model MTRL . . . 211
6.5.2 Online MTRL . . . 212
6.5.3 MTRL with Infinite TTs . . . 213
6.5.4 Robustness to Hyper-parameters . . . 213
6.6 Conclusion . . . 214
6.7 Supplemental Materials: Proofs and Additional Details . . . 215
6.7.1 Theoretical Analysis and Proofs . . . 215
6.7.2 Experiment Settings and Additional Results . . . 224
6.7.3 Discussion: Potential Extension to Deep RL . . . 227

Chapter 7: Adapting to Novel Observation Feature Spaces 230
7.1 Introduction . . . 230
7.2 Related Work . . . 233
7.3 Background and Problem Setup . . . 235
7.3.1 Transfer Across Different Observation Spaces . . . 235
7.4 Proposed Approach . . . 236
7.4.1 Characterizing Conditions for Good Representations . . . 237
7.4.2 Algorithm: Learning and Transferring Model-based Regularizer . . . 241
7.4.3 Theoretical Analysis: Benefits of Transferable Dynamics Model . . . 243
7.5 Empirical Results . . . 247
7.5.1 Experiment Setting . . . 248
7.5.2 Experiment Results . . . 250
7.5.3 Ablation Study and Discussion . . . 250
7.6 Conclusion . . . 252

7.7 Supplemental Materials: Proofs and Additional Details . . . 253
7.7.1 Theoretical Analysis and Proofs . . . 253
7.7.2 Experiment Setting Details . . . 259

Chapter 8: Adapting to Various Tasks with A Single Foundation Model 266
8.1 Introduction . . . 266
8.2 Related Work . . . 268
8.3 Background and Problem Setup . . . 270
8.4 Proposed Approach . . . 272
8.4.1 Approach Overview and Model Architecture . . . 273
8.4.2 Control-centric Self-supervised Pretraining Objectives . . . 275
8.5 Empirical Results . . . 278
8.5.1 Experimental Setup . . . 278
8.5.2 Experimental Results . . . 281
8.5.3 Ablation and Discussion . . . 283
8.6 Conclusion . . . 285
8.7 Supplemental Materials: Additional Details and Experiments . . . 286
8.7.1 Experiment Setting Details . . . 286
8.7.2 Additional Experiment Results . . . 288

Chapter 9: Conclusions and Future Perspectives 291
9.1 Dissertation Summary . . . 291
9.2 Future Perspectives . . . 293

Bibliography 295

List of Tables

2.1 Comparison between our PA-AD and other evasion attack methods in Atari environments. . . . 23
2.2 Comparison between our PA-AD and other evasion attack methods in MuJoCo environments. . . . 23
2.3 Comparison between our PA-ATLA-PPO and other robust RL training methods in MuJoCo environments. . . . 24
2.4 Performance of PA-AD across difference choices of the relaxation hyperparameter λ . . . 64
2.5 Average training time of SA-RL and PA-AD in MuJoCo environments. . . . 66
2.6 Our PA-ATLA-PPO robust training method is more efficient than prior methods. . 69
2.7 Attacking robustly trained models in Atari environments. . . . 71
2.8 Comparison between our PA-ATLA-A2C and other robust RL training methods in Atari environments. . . . 73

3.1 Robustness and high natural performance of WocaR-DQN. . . . 96
3.2 Full robustness results of WocaR-PPO. . . . 111
3.3 Robustness of WocaR-DQN on Atari games. . . . 113
3.4 Robustness of WocaR-A2C on Atari games. . . . 113
3.5 Robustness and efficiency of WocaR-PPO with fewer training steps. . . . 114
3.6 Efficiency comparison of state-of-the-art robust training methods and WocaR-PPO. 115

8.1 A comparison between pretraining and finetuning. . . . 272
8.2 A list of domains and tasks used in pretraining and finetuning. . . . 286

List of Figures

2.1 An example that a myopic adversary is not the strongest. . . . 7
2.2 Equivalence between evasion attacks and policy perturbations. . . . 12
2.3 The relations of state perturbation, policy perturbation, and value perturbation. . . . 14
2.4 An overview of PA-AD compared with a heuristic attacker and an end-to-end RL attacker. . . . 17
2.5 Two examples of the outermost boundary. . . . 43
2.6 Value space of an example MDP. . . . 44
2.7 A simple MDP where MinBest Attacker cannot find the optimal adversary for a given victim policy. . . . 52
2.8 A simple MDP where the first version of MaxWorst Attacker cannot find the optimal adversary for a given victim policy. . . . 54
2.9 A simple MDP where the second version of MaxWorst Attacker cannot find the optimal adversary for a given victim policy. . . . 56
2.10 Comparison of different attack methods against DQN and A2C victims in Atari w.r.t. different budget ϵ's. . . . 63
2.11 Comparison of different attack methods against PPO victims in MuJoCo w.r.t. different budget ϵ's. . . . 63
2.12 Comparison of convergence rate between SA-RL and PA-AD in Ant and Cartpole. Results are averaged over 10 random seeds. . . . 65
2.13 Histograms of victim rewards under different hyperparameter settings of SA-RL and PA-AD on Walker. . . . 67

3.1 Policies have different vulnerabilities. . . . 74
3.2 Comparison between learned behaviors of prior methods and our robust training method. . . . 77
3.3 Geometric understanding of different training methods. . . . 81
3.4 Examples of state importance weight. . . . 89
3.5 Training architecture of WocaR-RL. . . . 90
3.6 Robustness, efficiency and high natural performance of WocaR-PPO . . . 94
3.7 Comparison between estimated worst-attack action values and actual worst-case rewards. . . . 97
3.8 Ablation study of WocaR-RL . . . 97
3.9 Comparisons under different attacks w.r.t. different budgets. . . . 114
3.10 Learning curves of natural rewards and worst-case rewards. . . . 116
3.11 Average natural rewards and worst-case rewards of WocaR-PPO with different κ_{wst} and other baselines. . . . 118
3.12 Ablation performance for the state importance weight. . . . 119

3.13 Ablation performance for the state regularization loss. . . . 119

4.1 An example of difficult poisoning. . . . 122
4.2 Online poisoning-learning process. . . . 126
4.3 Comparison of mean per-episode reward gained by VPG, PPO, A2C, ACKTR on various environments, under no poisoning, random poisoning, AC-P and VA2C-P. . . . 137
4.4 Extension of VA2C-P in variants scenarios. . . . 139
4.5 Different poison aims of poisoning in supervised learning and reinforcement learning. . . . 144
4.6 Additional Experimental Results. . . . 155
4.7 Poisoning off-policy algorithm SAC with VA2C-P. $\mathcal{D} = \mathcal{O}^r, C/K = 1, \epsilon = 0.6$. . . . 156

5.1 An example of test-time communication attacks in a communicative MARL system. 158
5.2 Rewards of our AME and baselines in all environments, under no attacker and varying numbers of adversaries for adaptive and various non-adaptive attacks. . . . 180
5.3 Hyper-parameter study of AME. . . . 182
5.4 Rewards of our AME and baselines in FoodCollector, under no attacker and varying numbers of adversaries for adaptive and heuristic (random message) attacks. . . . 189
5.5 Rewards of our AME and baselines in InventoryManager, under no attacker and varying numbers of adversaries for adaptive and various heuristic attacks. . . . 190
5.6 Precision and recall of MARL classification on MNIST without or with AME. . . . 191
5.7 Full results of hyper-parameter tests. . . . 191

6.1 An motivating example of similar state-action transition dynamics. . . . 193
6.2 An example of TTs. . . . 199
6.3 Performance of O-TempLe and FM-TempLe compared against state-of-the-art baselines. . . . 211
6.4 Hyper-parameter test of TempLe. . . . 214
6.5 Transfer learning with varying action size. . . . 225
6.6 More examples of the applicability of TempLe. . . . 226
6.7 Performance of O-TempLe on challenging single-task problems, compared with RMax and Q-learning. . . . 227
6.8 Extending TempLe to deep RL. . . . 229

7.1 An example of the transfer problem with changed observation space. . . . 231
7.2 The architecture of proposed transfer learning method. . . . 242
7.3 Our proposed transfer method outperforms all baselines in target tasks over all tested scenarios. . . . 249
7.4 Ablation study of our method on different transferred components. . . . 251
7.5 Sanity check verifies the effectiveness of our algorithm design. . . . 252
7.6 Hyper-parameter test of our method. . . . 252

8.1 Architecture of Control Transformer. . . . 274
8.2 The three terms of our proposed pretraining objective. . . . 276

8.3 Downstream learning rewards of SMART. 282
8.4 Downstream learning rewards in unseen tasks and domains of SMART. 282
8.5 Downstream learning rewards (normalized by expert score) of all methods. 283
8.6 Ablation study on proposed pretraining objective. 284
8.7 Ablation study of the effect of reward in pretraining and comparison of various model capacity. 284
8.8 Graphical relations of all tasks involved. 286
8.9 Downstream learning rewards of SMART using the Random pretraining dataset. . 289
8.10 Downstream learning rewards in unseen tasks and domains of SMART using the Random pretraining dataset. 289

List of Abbreviations

RL	Reinforcement Learning
SL	Supervised Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
FGSM	Fast Gradient Sign Method
PGD	Projected Gradient Decent

Chapter 1: Introduction

1.1 Background and Motivation

Empowered by high-performance computing techniques and large volumes of datasets, machine learning (ML) has achieved significant and rapid development in a wide range of applications in the past decade. In particular, reinforcement learning (RL) imitates the natural learning process of humans and can give birth to intelligent agents for automatic (sequential) decision-making. By learning from interactions and maximizing long-term utilities, RL agents are shown to achieve superhuman performance in many scenarios such as video games and board games [1, 2, 3]. However, the success of these learning-based intelligent agents is mainly evaluated in stable and stationary simulated environments, while there are still many challenges impeding the application of these intelligent agents in real-world scenarios.

We live in the era of information explosion, while *noisy, false, or even adversarial information* is mixed with useful information needed for making decisions. Recent work has revealed that a well-trained agent could greatly fail when the input is slightly but adversarially perturbed [4, 5]. That is, although an agent can learn to perform well in a clean environment, it may not make correct decisions in a noisy environment, making it risky to deploy these agents in high-stakes real world applications. On the other hand, the real world we live in is *ever-changing, where previous experience and methods are often challenged by new and unseen problems*. For example, the un-

precedented COVID-19 pandemic has caused dramatic changes of our lives in the past few years; everyone is more or less forced to adapt to a changed environment and adopt a new lifestyle. However, adapting to a new environment is not easy for intelligent agents as they require a large amount of data samples and computation to learn — it takes days if not months for face recognition systems to get used to masks on faces. Therefore, the ability of quickly and reliably adapt to changes in environment is critical for intelligent agents to make real-time decisions.

Towards building more trustworthy and more general intelligent systems, this dissertation aims to understand and improve the robustness and adaptability of decision-making agents. In particular, we focus on the challenging sequential decision making problem and the RL regime, and propose formulations and solutions centered on the following aspects.

• **Robustness.** The interactions between the agent and the environment are subject to perturbations. For instance, an autonomous driving system [6] may suffer from GPS signal interference or wind turbulence. Motivated by this practical challenge, this dissertation addresses the following questions. How vulnerable existing intelligent agents are against possibly noisy or even adversarially perturbed inputs [7, 8]? Can we make sure that the agent works as expected no matter how the input is perturbed [9, 10]?

• **Efficient Adaptability.** A trained agent could later face a shift of task specification (e.g., navigating to a new location) or changes of the underlying environment (e.g., navigating in a new city). In this case, traditional methods may discard the old model and train a new agent from scratch which is usually expensive and inefficient. On the contrary, this dissertation makes the agent adapt to the new setting with the main idea of "learning by analogy" [11]. More concretely, when the task or the environment gets changed, our goal is to let the agent efficiently adapt to the new environment by transferring pre-learned knowledge [12, 13, 14].

The above two aspects are essentially complementary: robustness focuses on the *external environmental shifts* where outside noise or adversaries perturb the agent-environment interactions, while adaptability emphasizes the *internal environmental shifts* that directly changes the structure of the underlying environments.

1.2 Common Preliminaries and Notations

Reinforcement Learning (RL). An RL environment is modeled by a Markov Decision Process (MDP), denoted by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a stochastic dynamics model[1], $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function and $\gamma \in [0, 1)$ is a discount factor. An agent takes actions based on a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. For any policy, its performance can be measured by the value function $V^\pi(s) := \mathbb{E}_{P,\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s]$, and the action value function $Q^\pi(s, a) := \mathbb{E}_{P,\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a]$. The general goal of RL is to learn a policy π^* that maximizes the policy value. In large-scale problems, we usually adopt Deep RL (DRL) methods, where the policy and its value functions can be parameterized and learned by neural networks. Widely-used DRL methods include DQN [15], PPO [16] and etc. In this dissertation, we aim to provide generic solutions to improve the robustness and adaptability of any existing RL algorithms, so that our proposed methods are usually agnostic to the base learning algorithm and serve as plug-in components.

[1] $\Delta(\mathcal{X})$ denotes the space of probability distributions over $\mathcal{X}$.

1.3 Organization and Overview

The first 4 chapters of this dissertation will introduce our efforts on understanding and improving the robustness of sequential decision-making systems from multiple aspects, including test-time adversarial attacks, training-time perturbations (poisoning attacks), single-agent scenarios and multi-agent scenarios. In Chapter 2, we propose to evaluate the worst-case performance and measure the vulnerability of well-trained RL agents by a provably optimal and efficient attack algorithm. Chapter 3 then presents a generic framework that trains robust RL agents and defend against a wide range of attackers. Chapter 2 and Chapter 3 both study the scenario where the training environment is safe, while the test-time (deployment) interaction with the environment is vulnerable to adversaries. But it is also possible that the training-time environment or interaction data is altered by malicious attackers, and this scenario is investigated in Chapter 4. The vulnerability to adversarial perturbations or unexpected inputs is not only a concern in a single-agent environment, but also a severe problem in multi-agent systems, where interaction between agents could render higher uncertainty and cause risks. Motivated by this, Chapter 5 shows how vulnerable the existing multi-agent systems are under untrustworthy communication between agents, and we present a simple yet effective algorithm that is guaranteed to be robust no matter how a subset of agents are malfunctioning. Our study reveals that existing RL learning algorithms are particularly vulnerable to adversarial attacks spanning various settings, and we have introduced a series of solutions to mitigate the vulnerability. Our proposed methods can significantly improve the robustness of RL agents against possible attacks, as suggested by both theory and experiments.

In the last 3 chapters, we present several methods that tackle challenging transfer learning

problems or multi-task learning problems. Chapter 6 focuses on adapting to unseen state-action pairs whose dynamics are similar to previously encountered ones, which achieves significant improvement on the overall sample efficiency in both theory and practice. Chapter 7 addresses a novel transfer learning scenario where a new task can have a totally different observation space compared to the old task (e.g., changing from LiDAR sensors to camera sensors), with a deep dive into the role of representation in reinforcement learning. The main idea of Chapter 7 is to utilize the underlying dynamics similarity between tasks to facilitate representation learning in the new task. Another study introduced in Chapter 8 takes an opposite perspective that utilizes the similarities of representation learning between tasks, and achieves fast adaptation to unseen tasks with different state spaces, action spaces and dynamics, by pre-training a foundation model with control relevant representation information. These proposed methods are essentially transferring knowledge from learned tasks to new tasks, a highly-desired property of intelligent agents.

Chapter 2: Measuring the Robustness of RL Policies

2.1 Introduction

Deep Reinforcement Learning (DRL) has achieved incredible success in many applications. However, recent works [4, 17] reveal that a well-trained RL agent may be vulnerable to test-time *evasion attacks*, making it risky to deploy RL models in high-stakes applications. As in most related works, we consider a *state adversary* which adds imperceptible noise to the observations of an agent such that its cumulative reward is reduced during test time.

In order to understand the vulnerability of an RL agent and to improve its certified robustness, it is important to evaluate the worst-case performance of the agent under any adversarial attacks with certain constraints. In other words, it is crucial to find the strongest/optimal adversary that can minimize the cumulative reward gained by the agent with fixed constraints, as motivated in [18]. Therefore, we focus on the following question:

Given an arbitrary attack radius (budget) ϵ for each step of the deployment, what is the worst-case performance of an agent under the strongest adversary?

Finding the strongest adversary in RL is challenging. Many existing attacks [4, 17] are based on heuristics, crafting adversarial states at every step independently, although steps are interrelated in contrast to image classification tasks. These heuristic methods can often effectively reduce the agent's reward, but are not guaranteed to achieve the strongest attack under a given

budget. This type of attack is "myopic" since it does not plan for the future. Figure 2.1 shows an intuitive example, where myopic adversaries only prevent the agent from selecting the best action in the current step, but the strongest adversary can strategically "lead" the agent to a trap, which is the worst event for the agent.

Achieving computational efficiency arises as another challenge in practice, even if the strongest adversary can be found in theory. A recent work [19] points out that learning the optimal state adversary is equivalent to learning an optimal policy in a new Markov Decision Process (MDP). A follow-up work [18] shows that the learned adversary significantly outperforms prior adversaries in MuJoCo games. However, the state space and the action space of the new MDP are both as large as the state space in the original environment, which can be high-dimensional in practice. For example, video games and autonomous driving systems use images as observations. In these tasks, learning the state adversary directly as in [18] becomes computationally intractable.

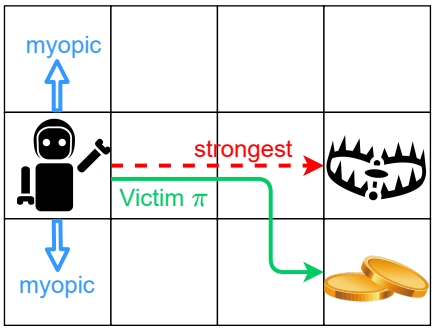


Figure 2.1: An example that a myopic adversary is not the strongest.

To overcome the above two challenges, **we propose a novel attack method called Policy Adversarial Actor Director (PA-AD)**, where we design a "director" and an "actor" that collaboratively finds the optimal state perturbations. In PA-AD, a director learns an MDP named *Policy Adversary MDP (PAMDP)*, and an actor is embedded in the dynamics of PAMDP. At each step, the director proposes a perturbing direction in the policy space, and the actor crafts a perturbation in the state space to lead the victim policy towards the proposed direction. Through a trail-and-error process, the director can find the optimal way to cooperate with the actor and attack the

victim policy. Theoretical analysis shows that the optimal policy in PAMDP induces an optimal state adversary. The size of PAMDP is generally smaller than the adversarial MDP defined by [18] and thus is easier to be learned efficiently using off-the-shelf RL algorithms. With our proposed *director-actor collaborative mechanism*, PA-AD outperforms state-of-the-art attacking methods on various types of environments, and improves the robustness of many DRL agents by adversarial training.

Summary of Contributions

(1) We establish a theoretical understanding of the optimality of evasion attacks from the perspective of policy perturbations, allowing a more efficient implementation of optimal attacks.

(2) We introduce a Policy Adversary MDP (PAMDP) model, whose optimal policy induces the optimal state adversary under any attacking budget ϵ.

(3) We propose a novel attack method, PA-AD, which efficiently searches for the optimal adversary in the PAMDP. PA-AD is a general method that works on stochastic and deterministic victim policies, vectorized and pixel state spaces, as well as discrete and continuous action spaces.

(4) Empirical study shows that PA-AD universally outperforms previous attacking methods in various environments, including Atari games and MuJoCo tasks. PA-AD achieves impressive attacking performance in many environments using very small attack budgets,

(5) Combining our strong attack PA-AD with adversarial training, we significantly improve the robustness of RL agents, and achieve the *state-of-the-art robustness in many tasks.*

2.2 Related Work

Heuristic-based Evasion Attacks on States There are many works considering evasion attacks on the state observations in RL. Huang et al. [4] first propose to use FGSM [20] to craft adversarial states such that the probability that the agent selects the "best" action is minimized. The same objective is also used in a recent work by [21], which adopts a Nesterov momentum-based optimization method to further improve the attack performance. Pattanaik et al. [17] propose to lead the agent to select the "worst" action based on the victim's Q function and use gradient descent to craft state perturbations. Zhang et al. [19] define the concept of a state-adversarial MDP (SAMDP) and propose two attack methods: Robust SARSA and Maximal Action Difference.

RL-based Evasion Attacks on States As discussed in Section 2.4.2, SA-RL [18] uses an end-to-end RL formulation to learn the optimal state adversary, which achieves state-of-the-art attacking performance in MuJoCo tasks. For a pixel state space, an end-to-end RL attacker may not work as shown by our experiment in Atari games (Section 2.5). Russo et al. [22] propose to use feature extraction to convert the pixel state space to a small state space and then learn an end-to-end RL attacker. But such feature extractions require expert knowledge and can be hard to obtain in many real-world applications. In contrast, our PA-AD works for both pixel and vector state spaces and does not require expert knowledge.

Other Works Related to Adversarial RL There are many other papers studying adversarial RL from different perspectives, including limited-steps attacking [23, 24], multi-agent scenarios [25], limited access to data [26], and etc. Adversarial action attacks [27, 28, 29, 30] are developed separately from state attacks; although we mainly consider state adversaries, our PA-AD can be extended to action attacks as formulated in Section 2.7.1. Poisoning [31, 32, 33, 34, 35]

is another type of adversarial attacks that manipulates the training data, different from evasion attacks that deprave a well-trained policy. Training a robust agent is the focus of many recent works [18, 19, 36, 37, 38, 39]. Although our main goal is to find a strong attacker, we also show by experiments that our proposed attack method significantly improves the robustness of RL agents by adversarial training.

2.3 Background and Problem Setup

The Victim RL Agent In RL, an agent interacts with an environment modeled by a Markov Decision Process (MDP) denoted as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is a state space with cardinality $|\mathcal{S}|$, $\mathcal{A}$ is an action space with cardinality $|\mathcal{A}|$, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function [1], $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. In this chapter, we consider a setting where the state space is much larger than the action space, which arises in a wide variety of environments. For notation simplicity, our theoretical analysis focuses on a finite MDP, but our algorithm applies to continuous state spaces and continuous action spaces, as verified in experiments. The agent takes actions according to its *policy*, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. We suppose the victim uses a fixed policy π with a function approximator (e.g. a neural network) during test time. We denote the *space of all policies* as Π, which is a Cartesian product of $|\mathcal{S}|$ simplices. The *value* of a policy $\pi \in \Pi$ for state $s \in \mathcal{S}$ is defined as $V^\pi(s) = \mathbb{E}_{\pi,P}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s]$.

Evasion Attacker Evasion attacks are test-time attacks that aim to reduce the expected total reward gained by the agent/victim. As in most literature [4, 17, 19], we assume the attacker knows the victim policy π (white-box attack). However, the attacker does not know the environ-

[1] $\Delta(X)$ denotes the the space of probability distributions over X.

ment dynamics, nor does it have the ability to change the environment directly. The attacker can observe the interactions between the victim agent and the environment, including states, actions and rewards. We focus on a typical *state adversary* [4, 19], which perturbs the state observations returned by the environment before the agent observes them. Note that the underlying states in the environment are not changed.

Formally, we model a state adversary by a function h which perturbs state $s \in \mathcal{S}$ into $\tilde{s} := h(s)$, so that the input to the agent's policy is $\tilde{s}$ instead of s. In practice, the adversarial perturbation is usually under certain constraints. In this chapter, we consider the common ℓ_p threat model [20]: $\tilde{s}$ should be in $\mathcal{B}_\epsilon(s)$, where $\mathcal{B}_\epsilon(s)$ denotes an ℓ_p norm ball centered at s with radius $\epsilon \geq 0$, a constant called the *budget* of the adversary for every step. With the budget constraint, we define the *admissible state adversary* and the *admissible adversary set* as below.

Definition 1 (**Set of Admissible State Adversaries** H_ϵ)**.** *A state adversary h is said to be admissible if $\forall s \in \mathcal{S}$, we have $h(s) \in \mathcal{B}_\epsilon(s)$. The set of all admissible state adversaries is denoted by H_ϵ.*

Then the goal of the attacker is to find an adversary h^* in H_ϵ that maximally reduces the cumulative reward of the agent. In this work, we propose a novel method to learn the optimal state adversary through the identification of an optimal *policy perturbation* defined and motivated in the next section.

2.4 Proposed Approach

2.4.1 Understanding Optimal Adversary via Policy Perturbations

In this section, we first motivate our idea of interpreting evasion attacks as perturbations of policies, then discuss how to efficiently find the optimal state adversary via the optimal policy perturbation.

Evasion Attacks Are Perturbations of Policies Although existing literature usually considers state-attacks and action-attacks separately, we point out that evasion attacks, either applied to states or actions, are essentially equivalent to perturbing the agent's policy π into another policy π_h in the policy space Π. For instance, as shown in Figure 2.2, if the adversary h alters state s into state $\tilde{s}$, the victim selects an action $\tilde{a}$ based on $\pi(\cdot|\tilde{s})$. This is equivalent to directly perturbing $\pi(\cdot|s)$ to $\pi_h(\cdot|s) := \pi(\cdot|\tilde{s})$. (See Section 2.7.1 for more detailed analysis including action adversaries.)

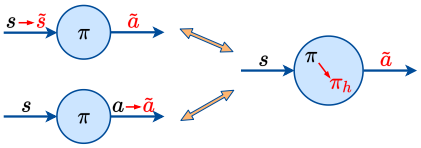


Figure 2.2: Equivalence between evasion attacks and policy perturbations.

In this chapter, we aim to find the optimal state adversary through the identification of the "optimal policy perturbation", which has the following **merits**. **(1)** $\pi_h(\cdot|s)$ usually lies in a lower dimensional space than $h(s)$ for an arbitrary state $s \in \mathcal{S}$. For example, in Atari games, the action space is discrete and small (e.g. $|\mathcal{A}| = 18$), while a state is a high-dimensional image. Then the state perturbation $h(s)$ is an image, while $\pi_h(\cdot|s)$ is a vector of size $|\mathcal{A}|$. **(2)** It is easier to characterize the optimality of a policy perturbation than a state perturbation. How a state perturbation changes the value of a victim policy depends on both the victim policy network and the environment dynamics. In contrast, how a policy perturbation changes the victim value only

depends on the environment. **(3)** Policy perturbation captures the essence of evasion attacks, and unifies state and action attacks. Although this chapter focuses on state-space adversaries, the learned "optimal policy perturbation" can also be used to conduct action-space attacks against the same victim.

Characterizing the Optimal Policy Adversary As depicted in Figure 2.3, the policy perturbation serves as a bridge connecting the perturbations in the state space and the value space. Our goal is to find the optimal state adversary by identifying the optimal "policy adversary". We first define an Admissible Adversarial Policy Set (Adv-policy-set) $\mathcal{B}_\epsilon^H(\pi) \subset \Pi$ as the set of policies perturbed from π by all admissible state adversaries $h \in H_\epsilon$. In other words, when a state adversary perturbs states within an ℓ_p norm ball $\mathcal{B}_\epsilon(\cdot)$, the victim policy is perturbed within $\mathcal{B}_\epsilon^H(\pi)$.

Definition 2 (Admissible Adversarial Policy Set (Adv-policy-set) $\mathcal{B}_\epsilon^H(\pi)$). *For an MDP $\mathcal{M}$, a fixed victim policy π, we define the admissible adversarial policy set (Adv-policy-set) w.r.t. π, denoted by $\mathcal{B}_\epsilon^H(\pi)$, as the set of policies that are perturbed from π by all admissible adversaries, i.e.,*

$$\mathcal{B}_\epsilon^H(\pi) := \{\pi_h \in \Pi : \exists h \in H_\epsilon \; s.t \; \forall s, \pi_h(\cdot|s) = \pi(\cdot|h(s))\}. \tag{2.1}$$

Remarks (1) $\mathcal{B}_\epsilon^H(\pi)$ is a subset of the policy space Π and it surrounds the victim π, as shown in Figure 2.3(middle). In the same MDP, $\mathcal{B}_\epsilon^H(\pi)$ varies for different victim π or different attack budget ϵ. (2) For a continuous function π (e.g., neural network), $\mathcal{B}_\epsilon^H(\pi)$ is connected and compact, and the value functions generated by all policies in the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$ form a polytope (Figure 2.3(right)), following the polytope theorem by Dadashi et al. [40].

Given that the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$ contains all the possible policies the victim may exe-

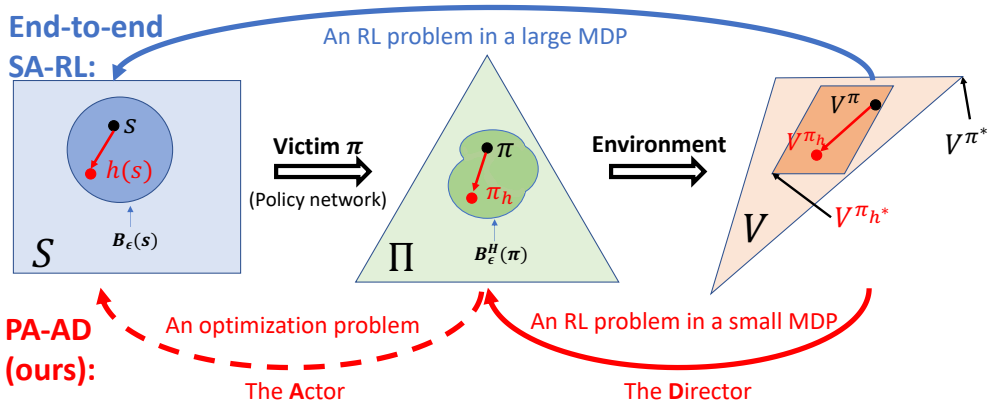


Figure 2.3: A state adversary h perturbs s into $h(s) \in \mathcal{B}_\epsilon(s)$ in the state space; hence, the victim's policy π is perturbed into π_h within the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$; as a result, the expected total reward the victim can gain becomes V^{π_h} instead of V^π.

cute under admissible state perturbations, we can characterize the optimality of a state adversary through the lens of policy perturbations. Recall that the attacker's goal is to find a state adversary $h^* \in H_\epsilon$ that minimizes the victim's expected total reward. From the perspective of policy perturbation, the attacker's goal is to perturb the victim's policy to another policy $\pi_{h^*} \in \mathcal{B}_\epsilon^H(\pi)$ with the lowest value. Therefore, we can define the optimal state adversary and the optimal policy adversary as below.

Definition 3 (**Optimal State Adversary** h^* **and Optimal Policy Adversary** π_{h^*})**.** *For an MDP* $\mathcal{M}$*, a fixed policy* π*, and an admissible adversary set* H_ϵ *with attacking budget* ϵ*,*

(1) an ***optimal state adversary*** h^* *satisfies* $h^* \in \operatorname{argmin}_{h \in H_\epsilon} V^{\pi_h}(s), \forall s \in \mathcal{S}$*, which leads to*

(2) an ***optimal policy adversary*** π_{h^*} *satisfies* $\pi_{h^*} \in \operatorname{argmin}_{\pi_h \in \mathcal{B}_\epsilon^H(\pi)} V^{\pi_h}(s), \forall s \in \mathcal{S}$*.*

Recall that π_h *is the perturbed policy caused by adversary* h*, i.e.,* $\pi_h(\cdot|s) = \pi(\cdot|h(s)), \forall s \in \mathcal{S}$*.*

Definition 3 implies an equivalent relationship between the optimal state adversary and the optimal policy adversary: an optimal state adversary leads to an optimal policy adversary, and any state adversary that leads to an optimal policy adversary is optimal. Theorem 19 in Section 2.7.6

shows that there always exists an optimal policy adversary for a fixed victim π, and learning the optimal policy adversary is an RL problem. (A similar result have been shown by [19] for the optimal state adversary, while we focus on the policy perturbation.) Due to the equivalence, if one finds an optimal policy adversary π_{h^*}, then the optimal state adversary can be found by executing targeted attacks with target policy π_{h^*}. However, directly finding the optimal policy adversary in the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$ is challenging since $\mathcal{B}_\epsilon^H(\pi)$ is generated by all admissible state adversaries in H_ϵ and is hard to compute.

To address this challenge, we first get insights from theoretical characterizations of the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$. Theorem 4 below shows that the "outermost boundary" of $\mathcal{B}_\epsilon^H(\pi)$ always contains an optimal policy adversary. Intuitively, a policy π' is in the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)$ if and only if no policy in $\mathcal{B}_\epsilon^H(\pi)$ is farer away from π than π' in the direction $\pi' - \pi$. Therefore, if an adversary can perturb a policy along a direction, it should push the policy as far away as possible in this direction under the budget constraints. Then, the adversary is guaranteed to find an optimal policy adversary after trying all the perturbing directions. In contrast, such a guarantee does not exist for state adversaries, justifying the benefits of considering policy adversaries. Our proposed algorithm in Section 2.4.2 applies this idea to find the optimal attack: *an RL-based director searches for the optimal perturbing direction, and an actor is responsible for pushing the policy to the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)$ with a given direction.*

Theorem 4. *For an MDP $\mathcal{M}$, a fixed policy π, and an admissible adversary set H_ϵ, define the* ***outermost boundary*** *of the admissible adversarial policy set $\mathcal{B}_\epsilon^H(\pi)$ w.r.t π as*

$$\partial_\pi \mathcal{B}_\epsilon^H(\pi) := \{\pi' \in \mathcal{B}_\epsilon^H(\pi) : \forall s \in \mathcal{S}, \theta > 0, \nexists \hat{\pi} \in \mathcal{B}_\epsilon^H(\pi) \text{ s.t. } \hat{\pi}(\cdot|s) = \pi'(\cdot|s) + \theta(\pi'(\cdot|s) - \pi(\cdot|s))\}. \tag{2.2}$$

Then there exists a policy $\tilde{\pi} \in \partial_\pi \mathcal{B}_\epsilon^H(\pi)$, such that $\tilde{\pi}$ is the optimal policy adversary w.r.t. π.

Theorem 4 is proven in Section 2.7.4.3, and we visualize the outermost boundary in Section 2.7.4.5.

2.4.2 PA-AD: Optimal and Efficient Evasion Attack

In this section, we first formally define the optimality of an attack algorithm and discuss some existing attack methods. Then, based on the theoretical insights in Section 2.4.1, we introduce our algorithm, *Policy Adversarial Actor Director (PA-AD)* that has an optimal formulation and is efficient to use.

Although many attack methods for RL agents have been proposed [4, 17, 19], it is not yet well-understood how to characterize the strength and the optimality of an attack method. Therefore, we propose to formulate the optimality of an attack algorithm, which answers the question "whether the attack objective finds the strongest adversary".

Definition 5 (Optimal Formulation of Attacking Algorithm)**.** *An attacking algorithm* Algo *is said to have an optimal formulation iff for any MDP $\mathcal{M}$, policy π and admissible adversary set H_ϵ under attacking budget ϵ, the set of optimal solutions to its objective, $H_\epsilon^{\mathsf{Algo}}$, is a subset of the optimal adversaries against π, i.e., $H_\epsilon^{\mathsf{Algo}} \subseteq H_\epsilon^* := \{h^* | h^* \in \operatorname{argmin}_{h \in H_\epsilon} V^{\pi_h}(s), \forall s \in \mathcal{S}\}$.*

Many heuristic-based attacks, although are empirically effective and efficient, do not meet the requirements of optimal formulation. In Section 2.7.7, we categorize existing heuristic attack methods into four types, and theoretically prove that there exist scenarios where these heuristic methods may not find the strongest adversary. A recent paper [18] proposes to learn the optimal state adversary using RL methods, which we will refer to as *SA-RL* in our paper for simplicity.

SA-RL can be viewed as an "end-to-end" RL attacker, as it directly learns the optimal state adversary such that the value of the victim policy is minimized. The formulation of SA-RL satisfies Definition 5 and thus is optimal. However, SA-RL learns an MDP whose state space and action space are both the same as the original state space. If the original state space is high-dimensional (e.g. images), learning a good policy in the adversary's MDP may become computationally intractable, as empirically shown in Section 2.5.

Can we address the optimal attacking problem in an efficient manner? SA-RL treats the victim and the environment together as a black box and directly learns a state adversary. But if the victim policy is known to the attacker (e.g. in adversarial training), we can exploit the victim model and simplify the attacking problem while maintaining the optimality. Therefore, we propose a novel algorithm, *Policy Adversarial Actor Director (PA-AD)*, that has optimal formulation and is generally more efficient than SA-RL. PA-AD decouples the whole attacking process into two simpler components: policy perturbation and state perturbation, solved by a "director" and an "actor" through collaboration. The director learns the optimal policy perturbing direction with

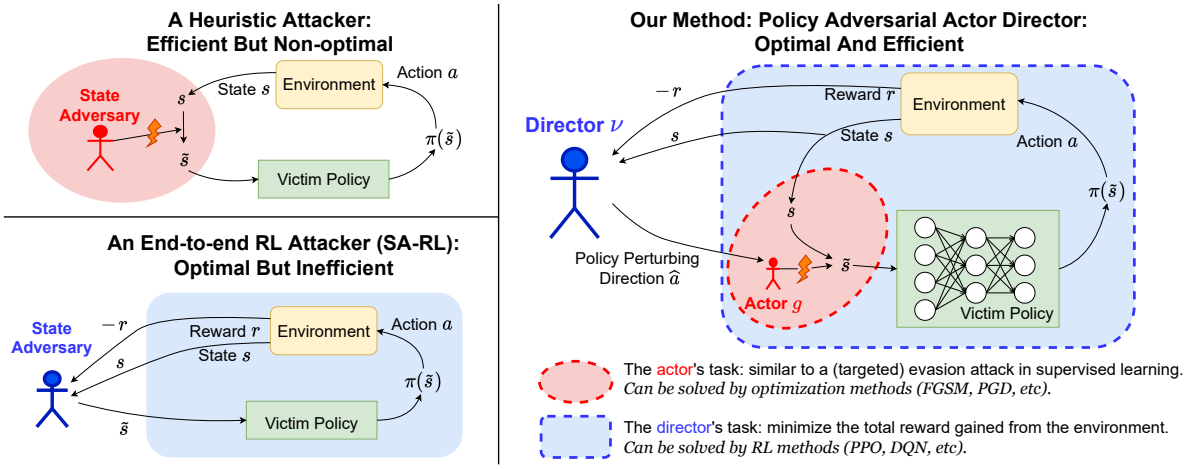


Figure 2.4: An overview of PA-AD compared with a heuristic attacker and an end-to-end RL attacker.

RL methods, while the actor crafts adversarial states at every step such that the victim policy is perturbed towards the given direction. Compared to the black-box SA-RL, PA-AD is a white-box attack, but works for a broader range of environments more efficiently. Note that PA-AD can be used to conduct black-box attack based on the transferability of adversarial attacks [4], although it is out of the scope of this chapter.

Formally, for a given victim policy π, our proposed PA-AD algorithm solves a *Policy Adversary MDP (PAMDP)* defined in Definition 6. An actor denoted by g is embedded in the dynamics of the PAMDP, and a director searches for an optimal policy ν^* in the PAMDP.

Definition 6 (Policy Adversary MDP (PAMDP) $\widehat{\mathcal{M}}$). *Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, a fixed **stochastic** victim policy π, an attack budget $\epsilon \geq 0$, we define a Policy Adversarial MDP $\widehat{\mathcal{M}} = \langle \mathcal{S}, \widehat{\mathcal{A}}, \widehat{P}, \widehat{R}, \gamma \rangle$, where the action space is $\widehat{\mathcal{A}} := \{d \in [-1,1]^{|\mathcal{A}|}, \sum_{i=1}^{|\mathcal{A}|} d_i = 0\}$, and $\forall s, s' \in \mathcal{S}, \forall \widehat{a} \in \widehat{\mathcal{A}}$,*

$$\widehat{P}(s'|s,\widehat{a}) = \sum\nolimits_{a \in \mathcal{A}} \pi(a|g(\widehat{a},s))P(s'|s,a), \quad \widehat{R}(s,\widehat{a}) = -\sum\nolimits_{a \in \mathcal{A}} \pi(a|g(\widehat{a},s))R(s,a),$$

where g is the actor function defined as

$$g(\widehat{a},s) = \text{argmax}_{\tilde{s} \in B_\epsilon(s)} \|\pi(\tilde{s}) - \pi(s)\| \text{ subject to } \big(\pi(\tilde{s}) - \pi(s)\big)^T \widehat{a} = \|\pi(\tilde{s}) - \pi(s)\| \|\widehat{a}\|. \quad (G)$$

*If the victim policy is **deterministic**, i.e., $\pi_D := \text{argmax}_a \pi(a|s)$, (subscript $_D$ stands for deterministic), the action space of PAMDP is $\widehat{\mathcal{A}}_D := \mathcal{A}$, and the actor function g_D is*

$$g_D(\widehat{a},s) = \text{argmax}_{\tilde{s} \in \mathcal{B}_\epsilon(s)} \big(\pi(\widehat{a}|\tilde{s}) - \max_{a \in \mathcal{A}, a \neq \widehat{a}} \pi(a|\tilde{s})\big). \quad (G_D)$$

Detailed definition of the deterministic-victim version of PAMDP is in Section 2.7.2.1.

A key to PA-AD is the director-actor collaboration mechanism. The input to director pol-

icy ν is the current state s in the original environment, while its output $\widehat{a}$ is a signal to the actor denoting "which direction to perturb the victim policy into". $\widehat{\mathcal{A}}$ is designed to contain all "perturbing directions" in the policy space. That is, $\forall \widehat{a} \in \widehat{A}$, there exists a constant $\theta_0 \geq 0$ such that $\forall \theta \leq \theta_0, \pi(\cdot|s) + \theta \frac{\widehat{a}}{\|\widehat{a}\|}$ belongs to the simplex $\Delta(A)$. The actor g takes in the state s and director's direction $\widehat{a}$ and then computes a state perturbation within the attack budget. Therefore, the director and the actor together induce a state adversary: $h(s) := g(\nu(s), s), \forall s \in \mathcal{S}$. The definition of PAMDP is slightly different for a stochastic victim policy and a deterministic victim policy, as described below.

For a stochastic victim π, the director's action $\widehat{a} \in \widehat{\mathcal{A}}$ is designed to be a unit vector lying in the policy simplex, denoting the perturbing direction in the policy space. The actor, once receiving the perturbing direction $\widehat{a}$, will "push" the policy as far as possible by perturbing s to $g(\widehat{a}, s) \in \mathcal{B}_\epsilon(s)$, as characterized by the optimization problem (G). In this way, the policy perturbation resulted by the director and the actor is always in the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)$ w.r.t. the victim π, where the optimal policy perturbation can be found according to Theorem 4.

For a deterministic victim π_D, the director's action $\widehat{a} \in \widehat{\mathcal{A}}_D$ can be viewed as a target action in the original action space, and the actor conducts targeted attacks to let the victim execute $\widehat{a}$, by forcing the logit corresponding to the target action to be larger than the logits of other actions.

In both the stochastic-victim and deterministic-victim case, PA-AD has an optimal formulation as stated in Theorem 7 (proven in Section 2.7.6.1).

Theorem 7 (Optimality of PA-AD). *For any MDP $\mathcal{M}$, any fixed victim policy π, and any attack budget $\epsilon \geq 0$, an optimal policy ν^* in $\widehat{\mathcal{M}}$ induces an optimal state adversary against π in $\mathcal{M}$. That is, the formulation of PA-AD is optimal, i.e., $H^{\text{PA-AD}} \subseteq H_\epsilon^*$.*

Efficiency of PA-AD As commonly known, the sample complexity and computational cost of learning an MDP usually grow with the cardinalities of its state space and action space. Both SA-RL and PA-AD have state space $\mathcal{S}$, the state space of the original MDP. But the action space of SA-RL is also $\mathcal{S}$, while our PA-AD has action space $\mathbb{R}^{|\mathcal{A}|}$ for stochastic victim policies, or $\mathcal{A}$ for deterministic victim policies. In most DRL applications, the state space (e.g., images) is much larger than the action space, then PA-AD is generally more efficient than SA-RL as it learns a smaller MDP.

The attacking procedure is illustrated in Algorithm 1. At step t, the director observes a state s_t, and proposes a policy perturbation $\widehat{a}_t$, then the actor searches for a state perturbation to meet the policy perturbation. Afterwards, the victim acts with the perturbed state $\tilde{s}_t$, then the director updates its policy based on the opposite value of the victim's reward. Note that the actor solves a constrained optimization problem, Equation (G_D) or Equation (G). Problem (G_D) is similar to a targeted attack in supervised learning, while the stochastic version (G) can be approximately solved with a Lagrangian relaxation. In Section 2.7.2.2, we provide our implementation details for solving the actor's optimization, which empirically achieves state-of-the-art attack performance as verified in Section 2.5.

Extending to Continuous Action Space Our PA-AD can be extended to environments

Algorithm 1: Policy Adversarial Actor Director (PA-AD)

Input: Initialization of director's policy ν; victim policy π; budget ϵ; start state s_0
for $t = 0, 1, 2, ...$ **do**
  Director samples a policy perturbing direction $\widehat{a}_t \sim \nu(\cdot|s_t)$
  Actor perturbs s_t to $\tilde{s}_t = g_D(\widehat{a}_t, s_t)$ if *Victim* is deterministic, otherwise to $\tilde{s}_t = g(\widehat{a}_t, s_t)$
  Victim takes action $a_t \sim \pi(\cdot|\tilde{s}_t)$, proceeds to s_{t+1}, receives r_t
  Director saves $(s_t, \widehat{a}_t, -r_t, s_{t+1})$ to its buffer
  Director updates its policy ν using any RL algorithm

with continuous action spaces, where the actor minimizes the distance between the policy action and the target action, i.e., $\operatorname{argmin}_{s' \in B_\epsilon(s)} \|\pi(s') - \widehat{a}\|$. More details and formal definitions of the variant of PA-AD in continuous action space are provided in Section 2.7.3. In Section 2.5, we show experimental results in MuJoCo tasks, which have continuous action spaces.

2.5 Empirical Results

In this section, we show that PA-AD produces stronger evasion attacks than state-of-the-art attack algorithms on various OpenAI Gym environments, including Atari and MuJoCo tasks. Also, our experiment justifies that PA-AD can evaluate and improve the robustness of RL agents.

Baselines and Performance Metric We compare our proposed attack algorithm with existing evasion attack methods, including *MinBest* [4] which minimizes the probability that the agent chooses the "best" action, *MinBest +Momentum* [21] which uses Nesterov momentum to improve the performance of MinBest, *MinQ* [17] which leads the agent to select actions with the lowest action values based on the agent's Q network, *Robust SARSA (RS)* [19] which performs the MinQ attack with a learned stable Q network, *MaxDiff* [19] which maximizes the KL-divergence between the original victim policy and the perturbed policy, as well as *SA-RL* [18] which directly learns the state adversary with RL methods. We consider state attacks with ℓ_∞ norm as in most literature [18, 19]. Section 2.7.8.1 provides hyperparameter settings and implementation details.

PA-AD Finds the Strongest Adversaries in Atari Games We first evaluate the performance of PA-AD against well-trained DQN [1] and A2C [41] victim agents on Atari games with pixel state spaces. The observed pixel values are normalized to the range of $[0, 1]$. SA-RL and PA-AD adversaries are learned using the ACKTR algorithm [42] with the same number of steps.

(Section 2.7.8.1 shows hyperparameter settings.) Table 2.1 presents the experiment results, where PA-AD significantly outperforms all baselines against both DQN and A2C victims. In contrast, SA-RL does not converge to a good adversary in the tested Atari games with the same number of training steps as PA-AD, implying the importance of sample efficiency. Surprisingly, using a relatively small attack budget ϵ, PA-AD leads the agent to the **lowest possible reward** in many environments such as Pong, RoadRunner and Tutankham, whereas other attackers may require larger attack budget to achieve the same attack strength. Therefore, we point out that *vanilla RL agents are extremely vulnerable to carefully learned adversarial attacks.* Even if an RL agent works well under naive attacks, a carefully learned adversary can let an agent totally fail with the same attack budget, which stresses the importance of evaluating and improving the robustness of RL agents using the strongest adversaries. In Section 2.7.8.2, we show more experiments with various selections of the budget ϵ, where one can see *PA-AD reduces the average reward more than all baselines over varying ϵ's in various environments.*

PA-AD Finds the Strongest Adversaries MuJoCo Tasks We further evaluate PA-AD on MuJoCo games, where both state spaces and action spaces are continuous. We use the same setting with [18], where both the victim and the adversary are trained with PPO [16]. During test time, the victim executes a deterministic policy, and we use the deterministic version of PA-AD with a continuous action space, as discussed in Section 2.4.2 and Section 2.7.3. We use the same attack budget ϵ as in [18] for all MuJoCo environments. Results in Table 2.2 show that PA-AD reduces the reward much more than heuristic methods, and also outperforms SA-RL in most cases. In Ant, our PA-AD achieves much stronger attacks than SA-RL, since PA-AD is more efficient than SA-RL when the state space is large. Admittedly, PA-AD requires additional knowledge of the victim model, while SA-RL works in a black-box setting. Therefore, SA-RL

	Environment	Natural Reward	ϵ	Random	MinBest	MinBest + Momentum	MinQ	MaxDiff	SA-RL	PA-AD (ours)
DQN	**Boxing**	96 ± 4	0.001	95 ± 4	53 ± 16	52 ± 18	88 ± 7	95 ± 5	94 ± 6	$\mathbf{19 \pm 11}$
	Pong	21 ± 0	0.0002	21 ± 0	-10 ± 4	-14 ± 2	14 ± 3	15 ± 4	20 ± 1	$\mathbf{-21 \pm 0}$
	RoadRunner	46278 ± 4447	0.0005	44725 ± 6614	17012 ± 6243	15823 ± 5252	5765 ± 12331	36074 ± 6544	43615 ± 7183	$\mathbf{0 \pm 0}$
	Freeway	34 ± 1	0.0003	34 ± 1	12 ± 1	12 ± 1	15 ± 2	22 ± 3	34 ± 1	$\mathbf{9 \pm 1}$
	Seaquest	10650 ± 2716	0.0005	8177 ± 2962	3820 ± 1947	2337 ± 862	6468 ± 2493	5718 ± 1884	8152 ± 3113	$\mathbf{2304 \pm 838}$
	Alien	1623 ± 252	0.00075	1650 ± 381	819 ± 486	775 ± 648	938 ± 446	869 ± 279	1693 ± 439	$\mathbf{256 \pm 210}$
	Tutankham	227 ± 29	0.00075	221 ± 65	30 ± 13	26 ± 16	88 ± 74	130 ± 48	202 ± 65	$\mathbf{0 \pm 0}$
A2C	**Breakout**	356 ± 79	0.0005	355 ± 79	86 ± 104	74 ± 95	N/A	304 ± 111	353 ± 79	$\mathbf{44 \pm 62}$
	Seaquest	1752 ± 70	0.005	1752 ± 73	356 ± 153	179 ± 83	N/A	46 ± 52	1752 ± 71	$\mathbf{4 \pm 13}$
	Pong	20 ± 1	0.0005	20 ± 1	-4 ± 8	-11 ± 7	N/A	18 ± 3	20 ± 1	$\mathbf{-13 \pm 6}$
	Alien	1615 ± 601	0.001	1629 ± 592	1062 ± 610	940 ± 565	N/A	1482 ± 633	1661 ± 625	$\mathbf{507 \pm 278}$
	Tutankham	258 ± 53	0.001	260 ± 54	139 ± 26	134 ± 28	N/A	196 ± 34	260 ± 54	$\mathbf{71 \pm 47}$
	RoadRunner	34367 ± 6355	0.002	35851 ± 6675	9198 ± 3814	5410 ± 3058	N/A	31856 ± 7125	36550 ± 6848	$\mathbf{2773 \pm 3468}$

Table 2.1: Average episode rewards $\pm$ standard deviation of vanilla DQN and A2C agents under different evasion attack methods in Atari environments. Results are averaged over 1000 episodes. Note that RS works for continuous action spaces, thus is not included. MinQ is not applicable to A2C which does not have a Q network. In each row, we bold the strongest (best) attack performance over all attacking methods.

is more applicable to black-box scenarios with a relatively small state space, whereas PA-AD is more applicable when the attacker has access to the victim (e.g. in adversarial training as shown in Table 2.3). Section 2.7.8.4 provides more empirical comparison between SA-RL and PA-AD, which shows that *PA-AD converges faster, takes less running time, and is less sensitive to hyperparameters than SA-RL* by a proper exploitation of the victim model.

Training and Evaluating Robust Agents A natural application of PA-AD is to evalu-

Environment	State Dimension	Natural Reward	ϵ	Random	MaxDiff	RS	SA-RL	PA-AD (ours)
Hopper	11	3167 ± 542	0.075	2101 ± 793	1410 ± 655	794 ± 238	636 ± 9	$\mathbf{160 \pm 136}$
Walker	17	4472 ± 635	0.05	3007 ± 1200	2869 ± 1271	1336 ± 654	1086 ± 516	$\mathbf{804 \pm 130}$
HalfCheetah	17	7117 ± 98	0.15	5486 ± 1378	1836 ± 866	489 ± 758	$\mathbf{-660 \pm 218}$	-356 ± 307
Ant	111	5687 ± 758	0.15	5261 ± 1005	1759 ± 828	268 ± 227	-872 ± 436	$\mathbf{-2580 \pm 872}$

Table 2.2: Average episode rewards $\pm$ standard deviation of vanilla PPO agent under different evasion attack methods in MuJoCo environments. Results are averaged over 50 episodes. Note that MinBest and MinQ do not fit this setting, since MinBest works for discrete action spaces, and MinQ requires the agent's Q network.

Environment	Model	Natural Reward	Random	MaxDiff	RS	SA-RL	PA-AD (ours)	Average reward across attacks
Hopper (state-dim: 11) ϵ: 0.075	SA-PPO	3705 ± 2	2710 ± 801	2652 ± 835	1130 ± 42	1076 ± 791	$\mathbf{856 \pm 21}$	1684.8
	ATLA-PPO	3291 ± 600	3165 ± 576	2814 ± 725	2244 ± 618	1772 ± 802	$\mathbf{1232 \pm 350}$	2245.4
	PA-ATLA-PPO (ours)	3449 ± 237	3325 ± 239	3145 ± 546	3002 ± 129	$\mathbf{1529 \pm 284}$	2521 ± 325	2704.4
Walker (state-dim: 17) ϵ: 0.05	SA-PPO	4487 ± 61	4867 ± 39	3668 ± 1789	3808 ± 138	2908 ± 1136	$\mathbf{1042 \pm 153}$	3258.6
	ATLA-PPO	3842 ± 475	3927 ± 368	3836 ± 492	3239 ± 894	3663 ± 707	$\mathbf{1224 \pm 770}$	3177.8
	PA-ATLA-PPO (ours)	4178 ± 529	4129 ± 78	4024 ± 572	3966 ± 307	3450 ± 478	$\mathbf{2248 \pm 131}$	3563.4
Halfcheetah (state-dim: 17) ϵ: 0.15	SA-PPO	3632 ± 20	3619 ± 18	3624 ± 23	3283 ± 20	3028 ± 23	$\mathbf{2512 \pm 16}$	3213.2
	ATLA-PPO	6157 ± 852	6164 ± 603	5790 ± 174	4806 ± 603	5058 ± 718	$\mathbf{2576 \pm 1548}$	4878.8
	PA-ATLA-PPO (ours)	6289 ± 342	6215 ± 346	5961 ± 53	5226 ± 114	4872 ± 79	$\mathbf{3840 \pm 673}$	5222.8
Ant (state-dim: 111) ϵ: 0.15	SA-PPO	4292 ± 384	4986 ± 452	4662 ± 522	3412 ± 1755	2511 ± 1117	$\mathbf{-1296 \pm 923}$	2855.0
	ATLA-PPO	5359 ± 153	5366 ± 104	5240 ± 170	4136 ± 149	3765 ± 101	$\mathbf{220 \pm 338}$	3745.4
	PA-ATLA-PPO (ours)	5469 ± 106	5496 ± 158	5328 ± 196	4124 ± 291	3694 ± 188	$\mathbf{2986 \pm 864}$	4325.6

Table 2.3: Average episode rewards $\pm$ standard deviation of robustly trained PPO agents under different attack methods. Results are averaged over 50 episodes. PA-ATLA-PPO is our robust training method. In each row corresponding to a robust agent, we bold the strongest attack. The gray cells are the most robust agents with the highest average rewards across attacks. Our PA-AD achieves the strongest attack against robust models, and our PA-ATLA-PPO achieves the most robust performance under multiple attacks. The attack budget ϵ's are the same as in [18].

ate the robustness of a known model, or to improve the robustness of an agent via adversarial training, where the attacker has white-box access to the victim. Inspired by ATLA [18] which alternately trains an agent and an SA-RL attacker, we propose PA-ATLA, which alternately trains an agent and a PA-AD attacker. In Table 2.3, we evaluate the performance of PA-ATLA for a PPO agent (namely PA-ATLA-PPO) in MuJoCo tasks, compared with state-of-the-art robust training methods, SA-PPO [19] and ATLA-PPO [18] [2]. From the table, we make the following observations. **(1)** *Our PA-AD attacker can significantly reduce the reward of previous "robust" agents.* Take the Ant environment as an example, although SA-PPO and ATLA-PPO agents gain 2k+ and 3k+ rewards respectively under SA-RL, the previously strongest attack, our PA-AD still reduces their rewards to about -1.3k and 200+ with the same attack budget. Therefore, we emphasize the importance of understanding the worst-case performance of RL agents, even robustly-trained

[2] We use **ATLA-PPO(LSTM)+SA Reg**, the most robust method reported by [18].

agents. **(2)** *Our PA-ATLA-PPO robust agents gain noticeably higher average rewards across attacks than other robust agents*, especially under the strongest PA-AD attack. Under the SA-RL attack, PA-ATLA-PPO achieves comparable performance with ATLA-PPO, although ATLA-PPO agents are trained to be robust against SA-RL. Due to the efficiency of PA-AD, PA-ATLA-PPO requires fewer training steps than ATLA-PPO, as justified in Section 2.7.8.5. The results of attacking and training robust models in Atari games are in Section 2.7.8.6 and Section 2.7.8.7, where PA-ATLA improves the robustness of Atari agents against strong attacks with ϵ as large as $3/255$.

2.6 Conclusion

This chapter introduces a novel attacking method to find the optimal attacks through collaboration between a designed function named "actor" and an RL-based learner named "director". The actor crafts state perturbations for a given policy perturbation direction, and the director learns to propose the best policy perturbation directions. Our proposed algorithm, PA-AD, is theoretically optimal and significantly more efficient than prior RL-based works in environments with large state spaces. Empirical results show that our proposed PA-AD universally outperforms state-of-the-art attacking methods in various Atari and MuJoCo environments. By applying PA-AD to adversarial training, we achieve state-of-the-art empirical robustness in multiple tasks under strong adversaries.

2.7 Supplemental Materials: Proofs and Additional Details

2.7.1 Relationship between Evasion Attacks and Policy Perturbations

As mentioned in Section 2.4.1, all evasion attacks can be regarded as perturbations in the policy space. To be more specific, we consider the following 3 cases, where we assume the victim uses policy π.

Case 1 (attack on states): define the state adversary as function h such that $\forall s \in \mathcal{S}$

$$h(s) = \tilde{s} \in \mathcal{B}_\epsilon(s) := \{s' \in \mathcal{S} : \|s' - s\| \leq \epsilon\}.$$

(For simplicity, we consider the attacks within a ϵ-radius norm ball.)

In this case, for all $s \in \mathcal{S}$, the victim samples action from $\pi_h(\cdot|s) = \pi(\cdot|h(s)) = \pi(\tilde{s})$, which is equivalent to the victim executing a perturbed policy $\pi_h \in \Pi$.

Case 2 (attack on actions for a deterministic π): define the action adversary as function $h^{(\mathcal{A})} : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$, and $\forall s \in \mathcal{S}, a \in \mathcal{A}$

$$h^{(\mathcal{A})}(a|s) = \tilde{a} \in \mathcal{B}_\epsilon(a) := \{a' \in \mathcal{A} : \|a' - a\| \leq \epsilon\}.$$

In this case, there exists a policy $\pi_{h^{(\mathcal{A})}}$ such that $\pi_{h^{(\mathcal{A})}}(s) = h^{(\mathcal{A})}(a|s) = \tilde{a}$, which is equivalent to the victim executing policy $\pi_{h^{(\mathcal{A})}} \in \Pi$.

Case 3 (attack on actions for a stochastic π): define the action adversary as function $h^{(\mathcal{A})}$:

$\mathcal{S} \times \mathcal{A} \to \mathcal{A}$, and $\forall s \in \mathcal{S}, a \in \mathcal{A}$

$$h^{(\mathcal{A})}(a|s) = \tilde{a} \text{ such that } \{\|\pi(\cdot|s) - Pr(\cdot|s)\| \leq \epsilon\},$$

where $Pr(\tilde{a}|s)$ denotes the probability that the action is perturbed into $\tilde{a}$.

In this case, there exists a policy $\pi_{h^{(\mathcal{A})}}$ such that $\pi_{h^{(\mathcal{A})}}(s) = Pr(\cdot|s)$, which is equivalent to the victim executing policy $\pi_{h^{(\mathcal{A})}} \in \Pi$.

Most existing evasion RL works [4, 17, 18, 19] focus on state attacks, while there are also some works [28, 29] studying action attacks. For example, Tessler et al. [29] consider Case 2 and Case 3 above and train an agent that is robust to action perturbations.

These prior works study either state attacks or action attacks, considering them in two different scenarios. However, the ultimate goal of robust RL is to train an RL agent that is robust to any threat models. Otherwise, an agent that is robust against state attacks may still be ruined by an action attacker. We take a step further to this ultimate goal by proposing a framework, policy attack, that unifies observation attacks and action attacks.

Although the focus of this chapter is on state attacks, we would like to point out that our proposed method can also deal with action attacks (the director proposes a policy perturbation direction, and an actor perturbs the action accordingly). It is also an exciting direction to explore hybrid attacks (multiple actors conducting states perturbations and action perturbations altogether, directed by a single director.) Our policy perturbation framework can also be easily incorporated in robust training procedures, as an agent that is robust to policy perturbations is simultaneously robust to both state attacks and action attacks.

2.7.2 Extentions and Additional Details of Our Algorithm

2.7.2.1 Attacking A Deterministic Victim Policy

For a deterministic victim $\pi_D = \text{argmax}_a \pi(a|s)$, we define Deterministic Policy Adversary MDP (D-PAMDP) as below, where a subscript $_D$ is added to all components to distinguish them from their stochastic counterparts. In D-PAMDP, the director proposes a target action $\widehat{a}_D \in \mathcal{A}(=: \widehat{\mathcal{A}}_D)$, and the actor tries its best to let the victim output this target action.

Definition 8 (Deterministic Policy Adversary MDP (D-PAMDP)). *Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, a fixed and deterministic victim policy π_D, we define a Deterministic Policy Adversarial MDP $\widehat{\mathcal{M}}_D = \langle \mathcal{S}, \widehat{\mathcal{A}}_D, \widehat{P}_D, \widehat{R}_D, \gamma \rangle$, where the action space is $\widehat{\mathcal{A}}_D = \widehat{\mathcal{A}}_D$, and $\forall s, s' \in \mathcal{S}, \forall \widehat{a} \in \mathcal{A}$,*

$$\widehat{P}_D(s'|s, \widehat{a}) = P(s'|s, \pi_D(g(\widehat{a}, s))), \quad \widehat{R}_D(s, \widehat{a}) = -R(s, \pi_D(g(\widehat{a}, s))).$$

The actor function g is defined as

$$g_D(\widehat{a}, s) = \text{argmax}_{\tilde{s} \in \mathcal{B}_\epsilon(s)} \big(\pi(\widehat{a}|\tilde{s}) - \max_{a \in \mathcal{A}, a \neq \widehat{a}} \pi(a|\tilde{s}) \big) \tag{G_D}$$

The optimal policy of D-PAMDP is an optimal adversary against π_D as proved in Section 2.7.6.1

2.7.2.2 Implementation Details of PA-AD

To address the actor function g (or g_D) defined in Equation (G) and (G_D), we let the actor maximize objectives J_D and J within the $\mathcal{B}_\epsilon(\cdot)$ ball around the original state, for a deterministic victim and a stochastic victim, respectively. Below we explicitly define J_D and J.

Actor Objective for Deterministic Victim For the deterministic variant of PA-AD, the actor function (G_D) is simple and can be directly solved to identify the optimal adversary. Concretely, we define the following objective

$$J_D(\tilde{s}; \widehat{a}, s) := \pi(\widehat{a}|\tilde{s}) - \max_{\mathrm{a}\in\mathcal{A}, \mathrm{a}\neq\widehat{\mathrm{a}}} \pi(a|\tilde{s}), \tag{J_D}$$

which can be realized with the multi-class classification hinge loss. In practice, a relaxed cross-entropy objective can also be used to maximize $\pi(\widehat{a}|\tilde{s})$.

Actor Objective for Stochastic Victim Different from the deterministic-victim case, the actor function for a stochastic victim defined in (G) requires solving a more complex optimization problem with a non-convex constraint set, which in practice can be relaxed to (J) (a Lagrangian relaxation) to efficiently get an approximation of the optimal adversary.

$$\mathrm{argmax}_{\tilde{s}\in\mathcal{B}_\epsilon(s)} J(\tilde{s}; \widehat{a}, s) := \|\pi(\cdot|\tilde{s}) - \pi(\cdot|s)\| + \lambda \times \mathsf{CosineSim}\big(\pi(\cdot|\tilde{s}) - \pi(\cdot|s),\ \widehat{a}\big) \tag{J}$$

where CosineSim in the second refers to the cosine similarity function; the first term measures how far away the policy is perturbed from the victim policy; λ is a hyper-parameter controlling the trade-off between the two terms. Experimental results show that our PA-AD is not sensitive to the value of λ. In our reported results in Section 2.5, we set λ as 1. Section 2.7.8.3 shows the evaluation of our algorithm using varying λ's.

Our implementation simply uses the Fast Gradient Sign Method (FGSM) [20] to approximately solve the actor's objective, although more advanced solvers such as Projected Gradient Decent (PGD) can be applied to further improve the performance. Experiment results in Section 2.5 verify that the above FGSM-based implementation achieves state-of-the-art attack performance.

What is the Influence of the Relaxation in (J)? First, it is important that the relaxation is only needed for a stochastic victim. For a deterministic victim, which is often the case in practice, the actor solves the original unrelaxed objective.

Second, as we will discuss in the next paragraph, the optimality of both SA-RL and PA-AD is regarding the formulation. That is, SA-RL and PA-AD formulate the optimal attack problem as an MDP whose optimal policy is the optimal adversary. However, in a large-scale task, deep RL algorithms themselves usually do not converge to the globally optimal policy and exploration becomes the main challenge. Thus, when the adversary's MDP is large, the suboptimality caused by the RL solver due to exploration difficulties could be much more severe than the suboptimality caused by the relaxation of the formulation. The comparison between SA-RL and PA-AD in our experiments can justify that the size of the adversary MDP has a larger impact than the relaxation of the problem on the final solution found by the attackers.

Optimality in Formulation v.s. Approximated Optimality in Practice PA-AD has an optimal formulation, as the optimal solution to its objective (the optimal policy in PAMDP) is always an optimal adversary (Theorem 7). Similarly, the previous attack method SA-RL has an optimal solution since the optimal policy in the adversary's MDP is also an optimal adversary. However, in practice where the environments are in a large scale and the number of samples is finite, the optimal policy is not guaranteed to be found by either PA-AD and SA-RL with deep RL algorithms. Therefore, for practical consideration, our goal is to search for a good solution or approximate the optimal solution using optimization techniques (e.g. actor-critic learning, one-step FGSM attack, Lagrangian relaxation for the stochastic-victim attack). In experiments (Section 2.5), we show that our implementation universally finds stronger attackers than prior methods, which verifies the effectiveness of both our theoretical framework and our practical

implementation.

2.7.3 Variants For Environments with Continuous Action Spaces

Although the analysis in the main paper focuses on an MDP whose action space is discrete, our algorithm also extends to a continuous action space as justified in our experiments.

2.7.3.1 For A Deterministic Victim

In this case, we can still use the formulation D-PAMDP, but a slightly different actor function

$$g_D(\widehat{a}, s) = \text{argmin}_{\tilde{s} \in \mathcal{B}_\epsilon(s)} \|\pi_D(\tilde{s}) - \widehat{a}\|. \tag{G_{CD}}$$

2.7.3.2 For A Stochastic Victim

Different from a stochastic victim in a discrete action space whose actions are sampled from a categorical distribution, a stochastic victim in a continuous action space usually follows a parametrized probability distribution with a certain family of distributions, usually Gaussian distributions. In this case, the formulation of PAMDP in Definition 6 is impractical. However, since the mean of a Gaussian distribution has the largest probability to be selected, one can still use the formulation in (G_{CD}), while replacing $\pi_D(\tilde{s})$ with the mean of the output distribution. Then, the director and the actor can collaboratively let the victim output a Gaussian distribution whose mean is the target action. If higher accuracy is needed, we can use another variant of PAMDP, named Continuous Policy Adversary MDP (C-PAMDP) that can also control the variance of the Gaussian distribution.

Definition 9 (Continuous Policy Adversary MDP (C-PAMDP)). *Given an MDP* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ *where* $\mathcal{A}$ *is continuous, a fixed and stochastic victim policy* π*, we define a Continuous Policy Adversarial MDP* $\widehat{\mathcal{M}}_C = \langle \mathcal{S}, \widehat{\mathcal{A}}_C, \widehat{P}_C, \widehat{R}_C, \gamma \rangle$*, where the action space is* $\widehat{\mathcal{A}}_D = \mathcal{A}$*, and* $\forall s, s' \in \mathcal{S}, \forall \widehat{a} \in \mathcal{A}$*,*

$$\widehat{P}(s'|s,\widehat{a}) = \int_{\mathcal{A}} \pi(a|g(\widehat{a},s))P(s'|s,a)\,da, \quad \widehat{R}(s,\widehat{a}) = -\int_{\mathcal{A}} \pi(a|g(\widehat{a},s))R(s,a)da.$$

The actor function g *is defined as*

$$g(\widehat{a},s) = \text{argmin}_{\tilde{s}\in\mathcal{B}_\epsilon(s)} \mathsf{KL}(\pi(\cdot|\tilde{s})||\mathcal{N}(\widehat{a},\sigma^2 I_{|\mathcal{A}|})). \tag{G_C}$$

where σ *is a hyper-parameter, and* $\mathcal{N}$ *denotes a multivariate Gaussian distribution.*

In short, Equation (G_C) encourages the victim to output a distribution that is similar to the target distribution. The hyperparameter σ controls the standard deviation of the target distribution. One can set σ to be small in order to let the victim execute the target action $\widehat{a}$ with higher probabilities.

2.7.4 Topological Properties of the Admissible Adversarial Policy Set

As discussed in Section 2.4.1, finding the optimal state adversary in the admissible adversary set H_ϵ can be converted to a problem of finding the optimal policy adversary in the Adv-policy-set $\mathcal{B}^H_\epsilon(\pi)$. In this section, we characterize the topological properties of $\mathcal{B}^H_\epsilon(\pi)$, and identify how the value function changes as the policy changes within $\mathcal{B}^H_\epsilon(\pi)$.

In Section 2.7.4.1, we show that under the settings we consider, $\mathcal{B}^H_\epsilon(\pi)$ is a connected and compact subset of Π. Then, Section 2.7.4.2, we define some additional concepts and re-formulate the notations. In Section 2.7.4.3, we prove Theorem 4 in Section 2.4.1 that the outermost boundary of $\mathcal{B}^H_\epsilon(\pi)$ always contains an optimal policy perturbation. In Section 2.7.4.4, we prove that

the value functions of policies in $\mathcal{B}_\epsilon^H(\pi)$ (or more generally, any connected and compact subset of Π) form a polytope. Section 2.7.4.6 shows an example of the polytope result with a 2-state MDP, and Section 2.7.4.5 shows examples of the outermost boundary defined in Theorem 4.

2.7.4.1 The Shape of Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$

It is important to note that $\mathcal{B}_\epsilon^H(\pi)$ is generally connected and compact as stated in the following lemma.

Lemma 10 (**$\mathcal{B}_\epsilon^H(\pi)$ is connected and compact**). *Given an MDP $\mathcal{M}$, a policy π that is a continuous mapping, and admissible adversary set $H_\epsilon := \{h : h(s) \in \mathcal{B}_\epsilon(s), \forall s \in \mathcal{S}\}$ (where $\epsilon > 0$ is a constant), the admissible adversarial policy set $\mathcal{B}_\epsilon^H(\pi)$ is a connected and compact subset of Π.*

Proof of Lemma 10. For an arbitrary state $s \in \mathcal{S}$, an admissible adversary $h \in H_\epsilon$ perturbs it within an ℓ_p norm ball $\mathcal{B}_\epsilon(s)$, which is connected and compact. Since π is a continuous mapping, we know $\pi(s)$ is compact and connected.

Therefore, $\mathcal{B}_\epsilon^H(\pi)$ as a Cartesian product of a finite number of compact and connected sets, is compact and connected. □

2.7.4.2 Additional Notations and Definitions for Proofs

We first formally define some concepts and notations.

For a stationary and stochastic policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, we can define the state-to-state transition function as

$$P^\pi(s'|s) := \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s,a), \forall s, s' \in \mathcal{S},$$

and the state reward function as

$$R^{\pi}(s) := \sum_{a \in \mathcal{A}} \pi(a|s) R(s,a), \forall s \in \mathcal{S}.$$

Then the value of π, denoted as V^{π}, can be computed via the Bellman equation

$$V^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi} = (I - \gamma P^{\pi})^{-1} R^{\pi}.$$

We further use Π_{s_i} to denote the projection of Π into the simplex of the i-th state, i.e., the space of action distributions at state s_i.

Let $f_v : \Pi \to \mathbb{R}^{|\mathcal{S}|}$ be a mapping that maps policies to their corresponding value functions. Let $\mathcal{V} = f_v(\Pi)$ be the space of all value functions.

Dadashi et al. [40] show that the image of f_v applied to the space of policies, i.e., $f_v(\Pi)$, form a (possibly non-convex) polytope as defined below.

Definition 11 (**(Possibly non-convex) polytope**). *A is called a **convex polytope** iff there are $k \in \mathbb{N}$ points $x_1, x_2, \cdots, x_k \in \mathbb{R}^n$ such that $A = Conv(x_1, \cdots, x_k)$. Furthermore, a **(possibly non-convex) polytope** is defined as a finite union of convex polytopes.*

And a more general concept is (possibly non-convex) polyhedron, which might not be bounded.

Definition 12 (**(Possibly non-convex) polyhedron**). *A is called a **convex polyhedron** iff it is the intersection of $k \in \mathbb{N}$ half-spaces $\hat{B}_1, \hat{B}_2, \cdots, \hat{B}_k$, i.e., $A = \cap_{i=1}^{k} \hat{B}_i$. Furthermore, a **(possibly non-convex) polyhedron** is defined as a finite union of convex polyhedra.*

In addition, let $Y^{\pi}_{s_1, \cdots, s_k}$ be the set of policies that agree with π on states $s_1, \cdots, s_k$. Dadashi et al. [40] also prove that the values of policies that agree on all but one state s, i.e., $f_v(Y^{\pi}_{\mathcal{S} \setminus \{s\}})$, form a line segment, which can be bracketed by two policies that are deterministic on s. Our

Lemma 16 extends this line segment result to our setting where policies are restricted in a subset of policies.

2.7.4.3 Proof of Theorem 4: Boundary Contains Optimal Policy Perturbations

Lemma 4 in [40] shows that policies agreeing on all but one state have certain monotone relations. We restate this result in Lemma 13 below.

Lemma 13 (Monotone Policy Interpolation)**.** *For any $\pi_0, \pi_1 \in Y^{\pi}_{S\setminus\{s\}}$ that agree with π on all states except for $s \in \mathcal{S}$, define a function $l : [0,1] \to \mathcal{V}$ as*

$$l(\alpha) = f_v(\alpha\pi_1 + (1-\alpha)\pi_0).$$

Then we have

(1) $l(0) \succcurlyeq l(1)$ or $l(1) \succcurlyeq l(0)$ ($\succcurlyeq$ stands for element-wise greater than or equal to);

(2) If $l(0) = l(1)$, then $l(\alpha) = l(0), \forall \alpha \in [0,1]$;

(3) If $l(0) \neq l(1)$, then there is a strictly monotonic rational function $\rho : [0,1] \to \mathbb{R}$, such that $l(\alpha) = \rho(\alpha)l(1) + (1-\rho(\alpha))l(0)$.

More intuitively, Lemma 13 suggests that the value of $\pi^\alpha := \alpha\pi_1 + (1-\alpha)\pi_0$ changes (strictly) monotonically with α, unless the values of π_0, π_1 and π_α are all equal. With this result, we can proceed to prove Theorem 4.

Proof of Theorem 4. We will prove the theorem by contradiction.

Suppose there is a policy $\hat{\pi} \in \mathcal{B}^H_\epsilon(\pi)$ such that $\hat{\pi} \notin \partial_\pi \mathcal{B}^H_\epsilon(\pi)$ and $f_v(\hat{\pi}) = V^{\hat{\pi}} < V^{\tilde{\pi}}, \forall \tilde{\pi} \in \mathcal{B}^H_\epsilon(\pi)$, i.e., there is no optimal policy adversary on the outermost boundary of $\mathcal{B}^H_\epsilon(\pi)$.

Then according to the definition of $\partial_\pi \mathcal{B}^H_\epsilon(\pi)$, there exists at least one state $s \in \mathcal{S}$ such that we can find another policy $\pi' \in \mathcal{B}^H_\epsilon(\pi)$ agreeing with $\hat{\pi}$ on all states except for s, where $\pi'(s)$

satisfies

$$\hat{\pi}(\cdot|s) = \alpha\pi(\cdot|s) + (1-\alpha)\pi'(\cdot|s)$$

for some scalar $\alpha \in (0, 1)$.

Then by Lemma 13, either of the following happens:

(1) $f_v(\pi) \succ f_v(\hat{\pi}) \succ f_v(\pi')$.

(2) $f_v(\pi) = f_v(\hat{\pi}) = f_v(\pi')$;

Note that $f_v(\hat{\pi}) \succ f_v(\pi)$ is impossible because we have assumed $\hat{\pi}$ has the lowest value over all policies in $\mathcal{B}_\epsilon^H(\pi)$ including π.

If (1) is true, then π' is a better policy adversary than $\hat{\pi}$ in $\mathcal{B}_\epsilon^H(\pi)$, which contradicts with the assumption.

If (2) is true, then π' is another optimal policy adversary. By recursively applying the above process to π', we can finally find an optimal policy adversary on the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)$, which also contradicts with our assumption.

In summary, there is always an optimal policy adversary lying on the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)$.

□

2.7.4.4 Proof of Theorem 14: Values of Policies in Admissible Adversarial Policy Set Form a Polytope

We first present a theorem that describes the "shape" of the value functions generated by all admissible adversaries (admissible adversarial policies).

Theorem 14 (**Policy Perturbation Polytope**). *For a finite MDP $\mathcal{M}$, consider a policy π and an Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$. The space of values (a subspace of $\mathbb{R}^{|\mathcal{S}|}$) of all policies in $\mathcal{B}_\epsilon^H(\pi)$, denoted by $\mathcal{V}^{\mathcal{B}_\epsilon^H(\pi)}$, is a (possibly non-convex) polytope.*

In the remaining of this section, we prove a more general version of Theorem 14 as below.

Theorem 15 (**Policy Subset Polytope**). *For a finite MDP $\mathcal{M}$, consider a connected and compact subset of Π, denoted as $\mathcal{T}$. The space of values (a subspace of $\mathbb{R}^{|\mathcal{S}|}$) of all policies in $\mathcal{T}$, denoted by $\mathcal{V}^{\mathcal{T}}$, is a (possibly non-convex) polytope.*

According to Lemma 10, $\mathcal{B}_\epsilon^H(\pi)$ is a connected and compact subset of Π, thus Theorem 14 is a special case of Theorem 15.

Additional Notations To prove Theorem 15, we further define a variant of $Y^\pi_{s_1,\cdots,s_k}$ as $\mathcal{T}^\pi_{s_1,\cdots,s_k}$, which is the set of policies that are in $\mathcal{T}$ and agree with π on states $s_1,\cdots,s_k$, i.e.,

$$\mathcal{T}^\pi_{s_1,\cdots,s_k} := \{\pi' \in \mathcal{T} : \pi'(s_i) = \pi(s_i), \forall i = 1,\cdots,k\}.$$

Note that different from $\mathcal{B}_\epsilon^H(\pi)$, $\mathcal{T}$ is no longer restricted under an admissible adversary set and can be any connected and compact subset of Π.

The following lemma shows that the values of policies in $\mathcal{T}$ that agree on all but one state form a line segment.

Lemma 16. *For a policy $\pi \in \mathcal{T}$ and an arbitrary state $s \in \mathcal{S}$, there are two policies in $\partial_\pi \mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$, namely π_s^-, π_s^+, such that $\forall \pi' \in \mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$,*

$$f_v(\pi_s^-) \preccurlyeq f_v(\pi') \preccurlyeq f_v(\pi_s^+), \tag{2.3}$$

where $\preccurlyeq$ denotes element-wise less than or equal to (if $a \preccurlyeq b$, then $a_i \leq b_i$ for all index i). Moreover, the image of f_v restricted to $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$ is a line segment.

Proof of Lemma 16. Lemma 5 in [40] has shown that f_v is infinitely differentiable on Π, hence we know $f_v(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})$ is compact and connected. According to Lemma 4 in [40], for any two policies $\pi_1, \pi_2 \in Y^\pi_{\mathcal{S}\setminus\{s\}}$, either $f_v(\pi_1) \preccurlyeq f_v(\pi_2)$, or $f_v(\pi_2) \preccurlyeq f_v(\pi_1)$ (there exists a total order). The same property applies to $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$ since $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$ is a subset of $Y^\pi_{\mathcal{S}\setminus\{s\}}$.

Therefore, there exists π_s^- and π_s^+ that achieve the minimum and maximum over all policies in $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$. Next we show π_s^- and π_s^+ can be found on the outermost boundary of $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$.

Assume $\pi_s^+ \notin \partial_\pi \mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$, and for all $\tilde{\pi} \in \mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$, $f_v(\tilde{\pi}) \prec f_v(\pi_s^+)$. Then we can find another policy $\pi' \in \partial_\pi \mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$ such that $\pi_s^+ = \alpha\pi + (1-\alpha)\pi'$ for some scalar $\alpha \in (0,1)$. Then according to Lemma 13, $f_v(\pi') \succcurlyeq f_v(\pi_s^+)$, contradicting with the assumption. Therefore, one should be able to find a policy on the outermost boundary of $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$ whose value dominates all other policies. And similarly, we can also find π_s^- on $\partial_\pi \mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$.

Furthermore, $f_v(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})$ is a subset of $f_v(Y^\pi_{\mathcal{S}\setminus\{s\}})$ since $\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}}$ is a subset of $Y^\pi_{\mathcal{S}\setminus\{s\}}$. Given that $f_v(Y^\pi_{\mathcal{S}\setminus\{s\}})$ is a line segment, and $f_v(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})$ is connected, we can conclude that $f_v(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})$ is also a line segment.

□

Next, the following lemma shows that π_s^+ and π_s^- and their linear combinations can generate values that cover the set $f_v(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})$.

Lemma 17. *For a policy $\pi \in \mathcal{T}$, an arbitrary state $s \in \mathcal{S}$, and π_s^+, π_s^- defined in Lemma 16, the following three sets are equivalent:*

(1) $f_v(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})$;

(2) $f_v\big(closure(\mathcal{T}^\pi_{\mathcal{S}\setminus\{s\}})\big)$, where $closure(\cdot)$ is the convex closure of a set;

(3) $\{f_v(\alpha\pi_s^+ + (1-\alpha)\pi_s^-) | \alpha \in [0,1]\}$;

(4) $\{\alpha f_v(\pi_s^+) + (1-\alpha) f_v(\pi_s^-) | \alpha \in [0,1]\}$*;*

Proof of Lemma 17. We show the equivalence by showing (1) $\subseteq$ (4) $\subseteq$ (3) $\subseteq$ (2) $\subseteq$ (1) as below.

(2) $\subseteq$ (1): For any $\pi_1, \pi_2 \in \mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}}$, without loss of generality, suppose $f_v(\pi_1) \preccurlyeq f_v(\pi_2)$. According to Lemma 13, for any $\alpha \in [0,1]$, $f_v(\pi_1) \preccurlyeq \alpha\pi_1 + (1-\alpha)\pi_2 \preccurlyeq f_v(\pi_2)$. Therefore, any convex combinations of policies in $\mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}}$ has value that is in the range of $f_v(\mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}})$. So the values of policies in the convex closure of $\mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}}$ do not exceed $f_v(\mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}})$, i.e., (2) $\subseteq$ (1).

(3) $\subseteq$ (2): Based on the definition, $\alpha\pi_s^+ + (1-\alpha)\pi_s^- \in closure(\mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}})$, so (3) $\subseteq$ (2).

(4) $\subseteq$ (3): According to Lemma 13, there exists a strictly monotonic rational function $\rho : [0,1] \to \mathbb{R}$, such that

$$l(\alpha) = f_v(\alpha\pi_s^+ + (1-\alpha)\pi_s^-) = \rho(\alpha) f_v(\pi_s^+) + (1-\rho(\alpha)) f_v(\pi_s^-).$$

Therefore, due to intermediate value theorem, for $\alpha \in [0,1]$, $\rho(\alpha)$ takes all values from 0 to 1. So (4) = (3).

(1) $\subseteq$ (4): Lemma 16 shows that $f_v(\mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}})$ is a line segment bracketed by $f_v(\pi_s^+)$ and $f_v(\pi_s^-)$. Therefore, for any $\pi' \in \mathcal{T}^{\pi}_{\mathcal{S}\backslash\{s\}}$, its value is a convex combination of $f_v(\pi_s^+)$ and $f_v(\pi_s^-)$.

□

Next, we show that the relative boundary of the value space constrained to $\mathcal{T}^{\pi}_{s_1,\cdots,s_k}$ is covered by policies that dominate or are dominated in at least one state. The **relative interior** of set A in B is defined as the set of points in A that have a relative neighborhood in $A \cap B$, denoted as $\text{relint}_B A$. The **relative boundary** of set A in B, denoted as $\partial_B A$, is defined as the set of points in A that are not in the relative interior of A, i.e., $\partial_B A = A \backslash \text{relint}_B A$. When there is no ambiguity, we omit the subscript of ∂ to simplify notations.

In addition, we introduce another notation $F^{\pi}_{s_1,\cdots,s_k} := V^{\pi} + span(C^{\pi}_{k+1},\cdots,C^{\pi}_{|\mathcal{S}|})$, where C^{π}_i stands for the i-th column of the matrix $(I-\gamma P^{\pi})^{-1}$. Note that $F^{\pi}_{s_1,\cdots,s_k}$ is the same with $H^{\pi}_{s_1,\cdots,s_k}$ in Dadashi et al. [40], and we change H to F in order to distinguish from the admissible adversary set H_ϵ defined in our paper.

Lemma 18. *For a policy $\pi \in \mathcal{T}$, $k \leq |\mathcal{S}|$, and a set of policies $\mathcal{T}^{\pi}_{s_1,\cdots,s_k}$ that agree with π on $s_1,\cdots,s_k$ (perturb π only at $s_{k+1},\cdots,s_{|\mathcal{S}|}$), define $\mathcal{V}^t := f_v(\mathcal{T}^{\pi}_{s_1,\cdots,s_k})$. Define two sets of policies $X^+_s := \{\pi' \in \mathcal{T}^{\pi}_{s_1,\cdots,s_k} : \pi'(\cdot|s) = \pi^+_s(\cdot|s)\}$, and $X^-_s := \{\pi' \in \mathcal{T}^{\pi}_{s_1,\cdots,s_k} : \pi'(\cdot|s) = \pi^-_s(\cdot|s)\}$. We have that the relative boundary of $\mathcal{V}^t$ in $F^{\pi}_{s_1,\cdots,s_k}$ is included in the value functions spanned by policies in $\mathcal{T}^{\pi}_{s_1,\cdots,s_k} \cap (X^+_{s_j} \cup X^-_{s_j})$ for at least one $s \notin \{s_1,\cdots,s_k\}$, i.e.,*

$$\partial\mathcal{V}^t \subset \bigcup_{j=k+1}^{|\mathcal{S}|} f_v(\mathcal{T}^{\pi}_{s_1,\cdots,s_k} \cap (X^+_{s_j} \cup X^-_{s_j}))$$

Proof of Lemma 18. We first prove the following claim:

Claim 1: For a policy $\pi_0 \in \mathcal{T}^{\pi}_{s_1,\cdots,s_k}$, if $\forall j \in \{k+1,\cdots,|\mathcal{S}|\}$, $\nexists \pi' \in closure(\mathcal{T}^{\pi}_{s_1,\cdots,s_k}) \cap (X^+_{s_j} \cup X^-_{s_j})$ such that $f_v(\pi') = f_v(\pi_0)$, then $f_v(\pi_0)$ has a relative neighborhood in $\mathcal{V}^t \cap F^{\pi}_{s_1,\cdots,s_k}$.

First, based on Lemma 16 and Lemma 17, we can construct a policy $\hat{\pi} \in closure(\mathcal{T}^{\pi}_{s_1,\cdots,s_k})$ such that $f_v(\hat{\pi}) = f_v(\pi_0)$ through the following steps:

Algorithm 2: Constructing $\hat{\pi}$

Set $\pi^k = \pi_0$

for $j = k+1,\cdots,|\mathcal{S}|$ **do**

  Find $\pi^+_{s_j}, \pi^-_{s_j} \in \mathcal{T}^{\pi_{j-1}}_{\mathcal{S}\setminus\{s_j\}}$

  Find $\pi^j = \hat{\alpha}_j\pi^+_{s_j} + (1-\hat{\alpha}_j)\pi^-_{s_j}$ such that $f_v(\pi_j) = f_v(\pi_{j-1})$

Return $\hat{\pi} = \pi^{|\mathcal{S}|}$

Denote the concatenation of α_j's as a vector $\hat{\alpha} := [\hat{\alpha}_{k+1},\cdots,\hat{\alpha}_{|\mathcal{S}|}]$.

According to the assumption that $\forall j \in \{k+1,\cdots,|\mathcal{S}|\}$, $\nexists \pi' \in closure(\mathcal{T}^{\pi}_{s_1,\cdots,s_k}) \cap (X^+_{s_j} \cup$

$X_{s_j}^-$) such that $f_v(\pi') = f_v(\pi_0)$, we have $\hat{\alpha}_j \notin \{0,1\}, \forall j = k+1, \cdots, |\mathcal{S}|$. Then, define a function $\phi : (0,1)^{|\mathcal{S}|-k} \to \mathcal{V}^t$ such that

$$\phi(\alpha) = f_v(\pi_\alpha), \text{ where } \begin{cases} \pi_\alpha(\cdot|s_j) = \alpha\pi_{s_j}^+ + (1-\alpha)\pi_{s_j}^- & \text{if } j \in \{k+1, \cdots, |\mathcal{S}|\} \\ \pi_\alpha(\cdot|s_j) = \hat{\pi}(\cdot|s_j) & \text{otherwise} \end{cases}$$

Then we have that

1. ϕ is continuously differentiable.
2. $\phi(\hat{\alpha}) = f_v(\hat{\pi})$.
3. $\frac{\partial \phi}{\partial \alpha_j}$ is non-zero at $\hat{\alpha}$ (because of Lemma 13 (3)).
4. $\frac{\partial \phi}{\partial \alpha_j}$ is along the i-the column of $(I - \gamma P^{\hat{\pi}})^{-1}$ (see Lemma 3 in Dadashi et al. [40]).

Therefore, by the inverse theorem function, there is a neighborhood of $\phi(\alpha) = f_v(\hat{\pi})$ in the image space.

Now we have proved Claim 1. As a result, for any policy $\pi_0 \in \mathcal{T}^\pi_{s_1,\cdots,s_k}$, if $f_v(\pi_0)$ is in the relative boundary of $\mathcal{V}^t$ in $F^\pi_{s_1,\cdots,s_k}$, then $\exists j \in \{k+1, \cdots, |\mathcal{S}|\}, \pi' \in closure(\mathcal{T}^\pi_{s_1,\cdots,s_k}) \cap (X_{s_j}^+ \cup X_{s_j}^-)$ such that $f_v(\pi') = f_v(\pi_0)$. Based on Lemma 17, we can also find $\pi'' \in \mathcal{T}^\pi_{s_1,\cdots,s_k} \cap (X_{s_j}^+ \cup X_{s_j}^-)$ such that $f_v(\pi'') = f_v(\pi_0)$. So Lemma 18 holds.

□

Now, we are finally ready to prove Theorem 15.

Proof of Theorem 15. We will show that $\forall \{s_1, \cdots, s_k\} \subseteq \mathcal{S}$, the value $\mathcal{V}^t = f_v(\mathcal{T}^\pi_{s_1,\cdots,s_k})$ is a polytope.

We prove the above claim by induction on the cardinality of the number of states k. In the base case where $k = |\mathcal{S}|$, $\mathcal{V}^t = \{f_v(\pi)\}$ is a polytope.

Suppose the claim holds for $k+1$, then we show it also holds for k, i.e., for a policy $\pi \in \Pi$, the value of $\mathcal{T}^{\pi}_{s_1,\cdots,s_k} \subseteq Y^{\pi}_{s_1,\cdots,s_k} \subseteq \Pi$ for a polytope.

According to Lemma 18, we have

$$\partial \mathcal{V}^t \subset \bigcup_{j=k+1}^{|\mathcal{S}|} f_v(\mathcal{T}^{\pi}_{s_1,\cdots,s_k} \cap (X^+_{s_j} \cup X^-_{s_j})) = \bigcup_{j=k+1}^{|\mathcal{S}|} \mathcal{V}^t \cap (F^+_{s_j} \cup F^-_{s_j}))$$

where $\partial \mathcal{V}^t$ denotes the relative boundary of $\mathcal{V}^t$ in $F^{\pi}_{s_1,\cdots,s_k}$; $F^+_{s_j}$ and $F^-_{s_j}$ are two affine hyperplanes of $F^{\pi}_{s_1,\cdots,s_k}$, standing for the value space of policies that agree with $\pi^+_{s_j}$ and $\pi^-_{s_j}$ in state s_j respectively.

Then we can get

1. $\mathcal{V}^t = f_v(\mathcal{T}^{\pi}_{s_1,\cdots,s_k})$ is closed as $\mathcal{T}^{\pi}_{s_1,\cdots,s_k}$ is compact and f_v is continuous.
2. $\partial \mathcal{V}^t \subset \bigcup_{j=k+1}^{|\mathcal{S}|}(F^+_{s_j} \cup F^-_{s_j}))$, a finite number of affine hyperplanes in $F^{\pi}_{s_1,\cdots,s_k}$.
3. $\mathcal{V}^t \cap F^+_{s_j}$ (or $\mathcal{V}^t \cap F^-_{s_j}$) is a polyhedron by induction assumption.

Hence, based on Proposition 1 by Dadashi et al. [40], we get $\mathcal{V}^t$ is a polyhedron. Since $\mathcal{V}^t \subseteq \mathcal{V}$ is bounded, we can further conclude that $\mathcal{V}^t$ is a polytope.

Therefore, for an arbitrary connected and compact set of policies $\mathcal{T} \subseteq \Pi$, let $\pi \in \mathcal{T}$ be an arbitrary policy in $\mathcal{T}$, then $f_v(\mathcal{T}) = f_v(\mathcal{T}^{\pi}_{\emptyset})$ is a polytope.

□

2.7.4.5 Examples of the Outermost Boundary

See Figure 2.5 for examples of the outermost boundary for different $\mathcal{B}^H_{\epsilon}(\pi)$'s.

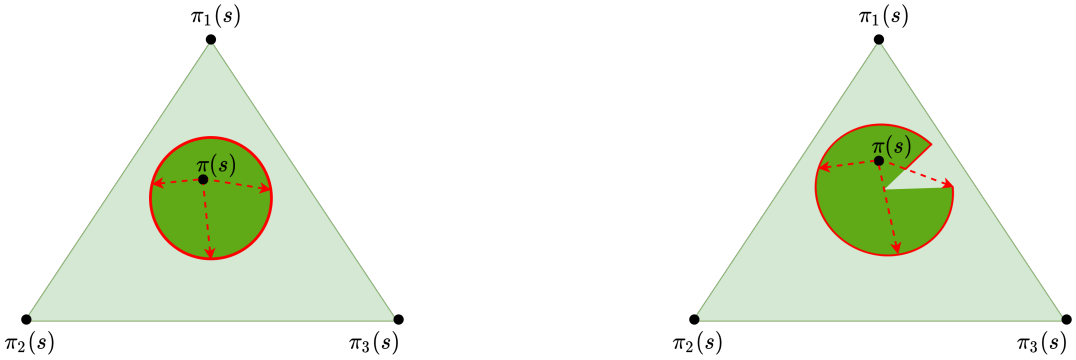


Figure 2.5: Two examples of the outermost boundary with $|\mathcal{A}| = 3$ actions at one single state s. The large triangle denotes the distributions over the action space at state s, i.e., Π_s; π_1, π_2 and π_3 are three policies that deterministically choose a_1, a_2 and a_3 respectively. π is the victim policy, the dark green area is the $\mathcal{B}_\epsilon^H(\pi)_s : \mathcal{B}_\epsilon^H(\pi) \cap \Pi_s$. The red solid curve depicts the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)_s$. Note that a policy is in the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)$ iff it is in the outermost boundary of $\mathcal{B}_\epsilon^H(\pi)_s$ for all $s \in \mathcal{S}$.

2.7.4.6 An Example of The Policy Perturbation Polytope

An example is given by Figure 2.6, where we define an MDP with 2 states and 3 actions. We train an DQN agent with one-hot encodings of the states, and then randomly perturb the states within an ℓ_∞ ball with $\epsilon = 0.8$. By sampling 5M random policies, and 100K random perturbations, we visualize the value space of approximately the whole policy space Π and the admissible adversarial policy set $\mathcal{B}_\epsilon^H(\pi)$, both of which are polytopes (boundaries are flat). A learning agent searches for the optimal policy π^* whose value is the upper right vertex of the larger blue polytope, while the attacker attempts to find an optimal adversary h^*, which perturbs a given clean policy π to the worst perturbed policy π_{h^*} whose value is the lower left vertex of the smaller green polytope. This also justifies the fact that learning an optimal adversary is as difficult as learning an optimal policy in an RL problem.

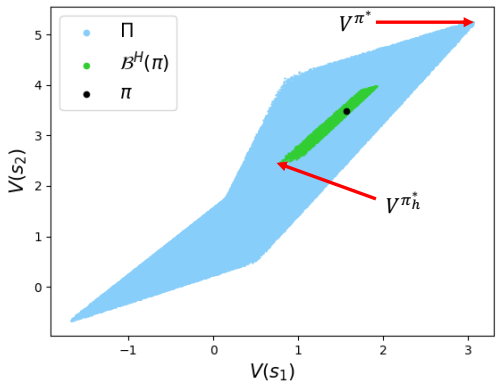


Figure 2.6: Value space of an example MDP. The values of the whole policy space Π form a polytope (blue) as suggested by [40]. The values of all perturbed policies with H_ϵ also form a polytope (green) as suggested by Theorem 14.

The example MDP $\mathcal{M}_{\text{ex}}$:

$$|\mathcal{A}| = 3, \gamma = 0.8$$

$$\hat{r} = [-0.1, -1., 0.1, 0.4, 1.5, 0.1]$$

$$\hat{P} = [[0.9, 0.1], [0.2, 0.8], [0.7, 0.3], [0.05, 0.95], [0.25, 0.75], [0.3, 0.7]]$$

The base/clean policy π:

$$\pi(a_1|s_1) = 0.215, \pi(a_2|s_1) = 0.429, \pi(a_3|s_1) = 0.356$$

$$\pi(a_1|s_2) = 0.271, \pi(a_2|s_2) = 0.592, \pi(a_3|s_2) = 0.137$$

2.7.5 Characterize Optimality of Evasion Attacks

In this section, we provide a detailed characterization for the optimality of evasion attacks from the perspective of policy perturbation, following Definition 5 in Section 2.4.2. Section 2.7.6 establishes the existence of the optimal policy adversary which is defined in Section 2.4.1. Section 2.7.6.1 then provides a proof for Theorem 7 that the formulation of PA-AD is optimal. We also analyze the optimality of heuristic attacks in Section 2.7.7.

2.7.6 Existence of An Optimal Policy Adversary

Theorem 19 (Existence of An Optimal Policy Adversary)**.** *Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, and a fixed stationary policy π on $\mathcal{M}$, let H_ϵ be a non-empty set of admissible state adversaries and $\mathcal{B}_\epsilon^H(\pi)$ be the corresponding Adv-policy-set, then there exists an optimal policy adversary $\pi_{h^*} \in \mathcal{B}_\epsilon^H(\pi)$ such that $\pi_{h^*} \in \operatorname{argmin}_{\pi_h \in \mathcal{B}_\epsilon^H(\pi)} V_{\mathcal{M}}^{\pi_h}(s), \forall s \in \mathcal{S}$.*

Proof. We prove Theorem 19 by constructing a new MDP corresponding to the original MDP $\mathcal{M}$ and the victim π.

Definition 20 (**Policy Perturbation MDP**)**.** *For a given MDP $\mathcal{M}$, a fixed stochastic victim policy π, and an admissible state adversary set H_ϵ, define a policy perturbation MDP as $\mathcal{M}_P = \langle \mathcal{S}, \mathcal{A}_P, P_P, R_P, \gamma \rangle$, where $\mathcal{A}_P = \Delta(\mathcal{A})$, and $\forall s \in \mathcal{S}, a_P \in \mathcal{A}_P$,*

$$R_P(s, a_P) := \begin{cases} -\sum_{a \in \mathcal{A}} a_P(a|s) R(s,a) & \text{if } \exists h \in H_\epsilon \text{ s.t. } a_P(\cdot|s) = \pi(\cdot|h(s)) \\ -\infty & \text{otherwise} \end{cases} \tag{2.4}$$

$$P_P(s'|s, a_P) := \sum_{a \in \mathcal{A}} a_P(a|s) P(s'|s,a) \tag{2.5}$$

Then we can prove Theorem 19 by proving the following lemma.

Lemma 21. *The optimal policy in $\mathcal{M}_P$ is an optimal policy adversary for π in $\mathcal{M}$.*

Let N_P denote the set of deterministic policies in $\mathcal{M}_P$. According to the traditional MDP theory [43], there exists a deterministic policy that is optimal in $\mathcal{M}_P$. Note that H_ϵ is non-empty, so there exists at least one policy in $\mathcal{M}_P$ with value $\geq -\infty$, and then the optimal policy should have value $\geq -\infty$. Denote this optimal and deterministic policy as $\nu_P^* \in N_P$. Then we write the

Bellman equation of ν_P^*, i.e.,

$$
\begin{aligned}
V_P^{\nu_P^*}(s) &= \max_{\nu_P \in N_P} R_P(s, \nu_P(s)) + \gamma \sum_{s' \in \mathcal{S}} P_P(s'|s, \nu_P(s)) V_P^{\nu_P}(s') \\
&= \max_{\nu_P \in N_P} \left[-\sum_{a \in \mathcal{A}} \nu_P(a|s) R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \nu_P(a|s) P(s'|s,a) V_P^{\nu_P}(s') \right] \qquad (2.6) \\
&= \max_{\nu_P \in N_P} \sum_{a \in \mathcal{A}} \nu_P(a|s) \left[-R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_P^{\nu_P}(s') \right]
\end{aligned}
$$

Note that $\nu_P^*(s)$ is a distribution on action space, $\nu_P^*(a|s)$ is the probability of a given by distribution $\nu^*(s)$.

Multiply both sides of Equation (2.6) by -1, and we obtain

$$
-V_P^{\nu_P^*}(s) = \min_{\nu \in N_P} \sum_{a \in \mathcal{A}} \nu_P(s)(a|s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \big(-V_P^{\nu_P}(s') \big) \right] \qquad (2.7)
$$

In the original MDP $\mathcal{M}$, an optimal policy adversary (if exists) π_{h^*} for π should satisfy

$$
V^{\pi_{h^*}}(s) = \min_{\pi_h \in \mathcal{B}_\epsilon^H(\pi)} \sum_{a \in \mathcal{A}} \pi_h(a|s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi_h}(s') \right] \qquad (2.8)
$$

By comparing Equation (2.7) and Equation (2.8) we get the conclusion that ν_P^* is an optimal policy adversary for π in $\mathcal{M}$.

□

2.7.6.1 Proof of Theorem 7: Optimality of Our PA-AD

In this section, we provide theoretical proof of the optimality of our proposed evasion RL algorithm PA-AD.

Optimality of PA-AD for A Stochastic Victim

We first build a connection between the PAMDP $\widehat{\mathcal{M}}$ defined in Definition 6 (Section 2.4.2)

and the policy perturbation MDP defined in Definition 20 (Section 2.7.6).

A deterministic policy ν in the PAMDP $\widehat{\mathcal{M}}$ can **induce** a policy $\hat{\nu}_P$ in $\mathcal{M}_P$ in the following way: $\widehat{\nu}_P(s) = \pi(\cdot|g(\nu(s), s)), \forall s \in \mathcal{S}$. More importantly, the values of ν and $\widehat{\nu}_P$ in $\widehat{\mathcal{M}}$ and $\mathcal{M}_P$ are equal because of the formulations of the two MDPs, i.e., $\widehat{V}^{\nu} = V_P^{\widehat{\nu}_P}$, where $\widehat{V}$ and V_P denote the value functions in $\widehat{\mathcal{M}}$ and V_P respectively.

Proposition 22 below builds the connection of the optimality between the policies in these two MDPs.

Proposition 22. *An optimal policy in $\widehat{\mathcal{M}}$ induces an optimal policy in $\mathcal{M}_P$.*

Proof of Proposition 22. Let ν^* be an deterministic optimal policy in $\widehat{\mathcal{M}}$, and it induces a policy in $\mathcal{M}_P$, namely $\widehat{\nu}_P$.

Let us assume $\widehat{\nu}_P$ is not an optimal policy in $\mathcal{M}_P$, hence there exists a policy ν_P^* in $\mathcal{M}_P$ s.t. $V_P^{\nu_P^*}(s) > V_P^{\widehat{\nu}_P}(s)$ for at least one $s \in \mathcal{S}$. And based on Theorem 4, we are able to find such a ν_P^* whose corresponding policy perturbation is on the outermost boundary of $\mathcal{B}(\pi)$, i.e., $\nu^* \in \partial_\pi \mathcal{B}_\epsilon^H(\pi)$.

Then we can construct a policy ν' in $\widehat{\mathcal{M}}$ such that $\nu'(s) = \nu_P^*(s) - \pi(s), \forall s \in \mathcal{S}$. And based on Equation (G), $\pi(\cdot|g(\nu'(s), s))$ is in $\partial_\pi \mathcal{B}(\pi(s))$ for all $s \in \mathcal{S}$. According to the definition of ∂_π, if two policy perturbations perturb π in the same direction and are both on the outermost boundary, then they are equal. Thus, we can conclude that $\pi(g(\nu'(s), s)) = \nu_P^*(s), \forall s \in \mathcal{S}$. Then we obtain $\widehat{V}^{\nu'}(s) = V_P^{\nu_P^*}(s), \forall s \in \mathcal{S}$.

Now we have conditions:

(1) $\widehat{V}^{\nu^*}(s) = V_P^{\widehat{\nu}_P}(s), \forall s \in \mathcal{S}$;

(2) $V_P^{\nu_P^*}(s) > V_P^{\widehat{\nu}_P}(s)$ for at least one $s \in \mathcal{S}$;

(3) $\exists \nu'$ such that $\widehat{V}^{\nu'}(s) = V_P^{\nu_P^*}(s), \forall s \in \mathcal{S}$.

From (1), (2) and (3), we can conclude that $\widehat{V}^{\nu'}(s) > \widehat{V}^{\nu^*}(s)$ for at least one $s \in \mathcal{S}$, which conflicts with the assumption that ν^* is optimal in $\widehat{\mathcal{M}}$. Therefore, Proposition 22 is proven.

□

Proposition 22 and Lemma 21 together justifies that the optimal policy of $\widehat{\mathcal{M}}$, namely ν^*, induces an optimal policy adversary for π in the original $\mathcal{M}$. Then, if the director learns the optimal policy in $\widehat{\mathcal{M}}$, then it collaborates with the actor and generates the optimal state adversary h^* by $h^*(s) = g(\nu^*(s), s), \forall s \in \mathcal{S}$.

Optimality of Our PA-AD for A Deterministic Victim

In this section, we show that the optimal policy in D-PAMDP (the deterministic variant of PAMDP defined in Section 2.7.2.1) also induces an optimal policy adversary in the original environment.

Let π_D be a deterministic policy reduced from a stochastic policy π, i.e.,

$$\pi_D(s) := \text{argmax}_{a \in \mathcal{A}} \pi(a|s), \forall s \in \mathcal{S}.$$

Note that in this case, the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi)$ is not connected as it contains only deterministic policies. Therefore, we re-formulate the policy perturbation MDP introduced in Section 2.7.6 with a deterministic victim as below:

Definition 23 (Deterministic Policy Perturbation MDP). *For a given MDP $\mathcal{M}$, a fixed deterministic victim policy π, and an admissible adversary set H_ϵ, define a deterministic policy pertur-*

bation MDP as $\mathcal{M}_{DP} = \langle \mathcal{S}, \mathcal{A}_{DP}, P_{DP}, R_{DP}, \gamma \rangle$*, where* $\mathcal{A}_{DP} = \mathcal{A}$*, and* $\forall s \in \mathcal{S}, a_{DP} \in \mathcal{A}_{DP}$*,*

$$R_{DP}(s, a_{DP}) := \begin{cases} -R(s, a_{DP}) & \text{if } \exists h \in H_\epsilon \text{ s.t. } a_{DP}(s) = \pi_D(h(s)) \\ -\infty & \text{otherwise} \end{cases} \tag{2.9}$$

$$P_{DP}(s'|s, a_{DP}) := P(s, a_{DP}) \tag{2.10}$$

$\mathcal{M}_{DP}$ can be viewed as a special case of $\mathcal{M}_P$ where only deterministic policies have $\geq -\infty$ values. Therefore Theorem 19 and Lemma 21 also hold for deterministic victims.

Next we will show that an optimal policy in $\widehat{\mathcal{M}}_D$ induces an optimal policy in $\mathcal{M}_{DP}$.

Proposition 24. *An optimal policy in* $\widehat{\mathcal{M}}_D$ *induces an optimal policy in* $\mathcal{M}_{DP}$*.*

Proof of Proposition 24. We will prove Proposition 24 by contradiction. Let ν^* be an optimal policy in $\widehat{\mathcal{M}}_D$, and it induces a policy in $\mathcal{M}_{DP}$, namely $\widehat{\nu}_{DP}$.

Let us assume $\widehat{\nu}_{DP}$ is not an optimal policy in $\mathcal{M}_{DP}$, hence there exists a deterministic policy ν^*_{DP} in $\mathcal{M}_{DP}$ s.t. $V_{DP}^{\nu^*_{DP}}(s) > V_{DP}^{\widehat{\nu}_{DP}}(s)$ for at least one $s \in \mathcal{S}$. Without loss of generality, suppose $V_{DP}^{\nu^*_{DP}}(s_0) > V_{DP}^{\widehat{\nu}_{DP}}(s_0)$.

Next we construct another policy ν' in $\widehat{\mathcal{M}}_D$ by setting $\nu'(s) = \nu^*_{DP}(s), \forall s \in \mathcal{S}$. Given that ν^*_{DP} is deterministic, ν' is also a deterministic policy. So we use $\nu^*_{DP}(s)$ and $\nu'(s)$ to denote the action selected by ν^*_{DP} and ν' respectively at state s.

For an arbitrary state s_i, let $a_i := \nu^*_{DP}(s_i)$. Since ν^*_{DP} is the optimal policy in $\mathcal{M}_{DP}$, we get that there exists a state adversary $h \in H_\epsilon$ such that $\pi_D(h(s_i)) = a_i$, or equivalently, there exists a state $\tilde{s}_i \in \mathcal{B}_\epsilon(s_i)$ such that $\text{argmax}_{a \in \mathcal{A}} \pi(\tilde{s}_i) = a_i$. Then, the solution to the actor's optimization problem (G_D) given direction a_i and state s_i, denoted as $\tilde{s}^*$, satisfies

$$\tilde{s}^* = \text{argmax}_{s' \in B_\epsilon(s)} \big(\pi(\widehat{a}|s') - \text{argmax}_{\mathrm{a} \in \mathcal{A}, \mathrm{a} \neq \widehat{\mathrm{a}}} \pi(a|s') \big) \tag{2.11}$$

and we can get

$$\pi(\widehat{a}|\tilde{s}^*) - \text{argmax}_{\text{a}\in\mathcal{A},\text{a}\neq\widehat{\text{a}}}\pi(a|\tilde{s}^*) \geq \pi(\widehat{a}|\tilde{s}_i) - \text{argmax}_{\text{a}\in\mathcal{A},\text{a}\neq\widehat{\text{a}}}\pi(a|\tilde{s}_i) > 0 \tag{2.12}$$

Given that $\text{argmax}_{a\in\mathcal{A}}\pi(a_i|\tilde{s}_i) = a_i$, we obtain $\text{argmax}_{a\in\mathcal{A}}\pi(a_i|\tilde{s}^*) = a_i$, and hence $\pi_D(g_D(a_i, s_i)) = a_i$. Since this relation holds for an arbitrary state s, we can get

$$\pi_D(g_D(\nu'(s), s)) = \pi_D(g_D(\nu'(s), s)) = \nu'(s), \forall s \in \mathcal{S} \tag{2.13}$$

Also, we have $\forall s \in \mathcal{S}$

$$\widehat{V}_D^{\nu'}(s) = \widehat{R}_D(s, \nu'(s)) + \sum_{s'\in\mathcal{S}} \widehat{P}_D(s'|s, \nu'(s))\widehat{V}_D^{\nu'}(s') \tag{2.14}$$

$$V_{DP}^{\nu^*_{DP}}(s) = R_{DP}(s, \nu^*_{DP}(s)) + \sum_{s'\in\mathcal{S}} P_{DP}(s'|s, \nu^*_{DP}(s))V_{DP}^{\nu^*_{DP}}((s') \tag{2.15}$$

Therefore, $\widehat{V}_D^{\nu'}(s) = V_{DP}^{\nu^*_{DP}}(s), \forall s \in \mathcal{S}$.

Then we have

$$\widehat{V}_D^{\nu'}(s_0) \leq \widehat{V}^{\nu^*}(s_0) = V_{DP}^{\widehat{\nu}_{DP}}(s_0) < V_{DP}^{\nu^*_{DP}}(s_0) = \widehat{V}_D^{\nu'}(s_0) \tag{2.16}$$

which gives $\widehat{V}_D^{\nu'}(s_0) < \widehat{V}_D^{\nu'}(s_0)$, so there is a contradiction.

□

Combining the results of Proposition 24 and Lemma 21 , for a deterministic victim, the optimal policy in D-PAMDP gives an optimal adversary for the victim.

2.7.7 Characterizing Optimality of Heuristic Attacks

There are many existing methods of finding adversarial state perturbations for a fixed RL policy, most of which are solving some optimization problems defined by heuristics. Although these methods are empirically shown to be effective in many environments, it is not clear how strong these adversaries are in general. In this section, we carefully summarize and categorize

existing heuristic attack methods into 4 types, and then characterize their optimality in theory.

2.7.7.1 *TYPE I* - Minimize The Best (MinBest)

A common idea of evasion attacks in supervised learning is to reduce the probability that the learner selects the "correct answer" [20]. Prior works [4, 21, 24] apply a similar idea to craft adversarial attacks in RL, where the objective is to minimize the probability of selecting the "best" action, i.e.,

$$h^{\text{MinBest}} \in \operatorname{argmin}_{h \in H_\epsilon} \pi_h(a^+|s), \forall s \in \mathcal{S} \tag{I}$$

where a^+ is the "best" action to select at state s. Huang et al.[4] define a^+ as $\operatorname{argmax}_{a \in \mathcal{A}} Q^\pi(s,a)$ for DQN, or $\operatorname{argmax}_{a \in \mathcal{A}} \pi(a|s)$ for TRPO and A3C with a stochastic π. Since the agent's policy π is usually well-trained in the original MDP, a^+ can be viewed as (approximately) the action taken by an optimal deterministic policy $\pi^*(s)$.

Lemma 25 (**Optimality of MinBest**). *Denote the set of optimal solutions to objective Equation* (I) *as* $H^{\textit{MinBest}}$. *There exist an MDP* $\mathcal{M}$ *and an agent policy* π, *such that* $H^{\textit{MinBest}}$ *does not contain an optimal adversary* h^*, *i.e.,* $H^{\textit{MinBest}} \cap H^*_\epsilon = \emptyset$.

Proof of Lemma 25. We prove this lemma by constructing the following MDP such that for any victim policy, there exists a reward configuration in which MinBest attacker is not optimal.

Here, let $r_1 = r(s_4|s_2,a_1), r_2 = r(s_5|s_2,a_2), r_3 = r(s_3|s_1,a_2)$. Assuming all the other rewards are zero, transition dynamics are deterministic, and states s_3, s_4, s_5 are the terminal states. For the sake of simplicity, we also assume that the discount factor here $\gamma = 1$.

Now given a policy π such that $\pi(a_1|s_1) = \beta_1$ and $\pi(a_1|s_2) = \beta_2$ ($\beta_1, \beta_2 \in [0,1]$), we could find

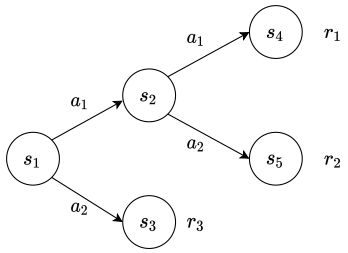


Figure 2.7: A simple MDP where MinBest Attacker cannot find the optimal adversary for a given victim policy.

r_1, r_2, r_3 such that the following constraints hold:

$$r_1 > r_2 \iff Q^\pi(s_1, a_1) > Q^\pi(s_1, a_2) \tag{2.17}$$

$$\beta_2 r_1 + (1-\beta_2) r_2 > r_3 \iff Q^\pi(s_2, a_1) > Q^\pi(s_2, a_2) \tag{2.18}$$

$$r_3 > (\beta_2 - \epsilon_2) r_2 + (1 - \beta_2 + \epsilon_2) r_2 \iff r_3 > Q^\pi(s_1, a_1) - \epsilon_2 (r_1 - r_2) \tag{2.19}$$

Now we consider the Adv-policy-set

$$\mathcal{B}_\epsilon^H(\pi) = \Big\{ \pi' \in \Pi \;\Big|\; \|\pi'(\cdot|s_1) - \pi(\cdot|s_1)\| < \epsilon_1, \|\pi'(\cdot|s_2) - \pi(\cdot|s_2)\| < \epsilon_2 \Big\}.$$

Under these three linear constraints, the policy given by MinBest attacker satisfies that $\pi_{h^{\text{MinBest}}}(a_1|s_1) = \beta_1 - \epsilon_1$, and $\pi_{h^{\text{MinBest}}}(a_1|s_2) = \beta_2 - \epsilon_2$. On the other hand, we can find another admissible policy adversary $\pi_{h^*}(a_1|s_1) = \beta_1 + \epsilon_1$, and $\pi_{h^*}(a_1|s_2) = \beta_2 - \epsilon_2$. Now we show that $V^{\pi_{h^*}}(s_1) < V^{\pi_{h^{\text{MinBest}}}}(s_1)$, and thus MinBest attacker is not optimal.

$$V^{\pi_{h^{\text{MinBest}}}}(s_1) = (\beta_1 - \epsilon_1)\Big[(\beta_2 - \epsilon_2) r_1 + (1 - \beta_2 + \epsilon_2) r_2\Big] + (1 - \beta_1 + \epsilon_1) r_3 \tag{2.20}$$

$$= (\beta_1 - \epsilon_1)(\beta_2 - \epsilon_2) r_1 + (\beta_1 - \epsilon_1)(1 - \beta_2 + \epsilon_2) r_2 + (1 - \beta_1 + \epsilon_1) r_3 \tag{2.21}$$

$$V^{\pi_{h^*}}(s_1) = (\beta_1 + \epsilon_1)\Big[(\beta_2 - \epsilon_2) r_1 + (1 - \beta_2 + \epsilon_2) r_2\Big] + (1 - \beta_1 - \epsilon_1) r_3 \tag{2.22}$$

$$= (\beta_1 + \epsilon_1)(\beta_2 - \epsilon_2) r_1 + (\beta_1 + \epsilon_1)(1 - \beta_2 + \epsilon_2) r_2 + (1 - \beta_1 - \epsilon_1) r_3 \tag{2.23}$$

Therefore,

$$V^{\pi_{h^*}}(s_1) - V^{\pi_{h^{\text{MinBest}}}}(s_1) = 2\epsilon_1(\beta_2 - \epsilon_2)r_2 + 2\epsilon_1(1 - \beta_2 + \epsilon_2)r_2 - 2\epsilon_1 r_3 \tag{2.24}$$

$$= 2\epsilon_1\Big[(\beta_2 - \epsilon_2)r_2 + (1 - \beta_2 + \epsilon_2)r_2 - r_3\Big] \tag{2.25}$$

$$< 0 \;\text{ Because of the constraint Equation (2.19)} \tag{2.26}$$

□

2.7.7.2 *TYPE II* - Maximize The Worst (MaxWorst)

Pattanaik et al. [17] point out that only preventing the agent from selecting the best action does not necessarily result in a low total reward. Instead, Pattanaik et al. [17] propose another objective function which maximizes the probability of selecting the worst action, i.e.,

$$h^{\text{MaxWorst}} \in \text{argmax}_{h \in H_\epsilon} \pi_h(a^-|s), \forall s \in \mathcal{S} \tag{II}$$

where a^- refers to the "worst" action at state s. Pattanaik et al.[17] define the "worst" action as the actions with the lowest Q value, which could be ambiguous, since the Q function is policy-dependent. If a worst policy $\pi^- \in \text{argmin}_\pi V^\pi(s), \forall s \in \mathcal{S}$ is available, one can use $a^- = \text{argmin} Q^{\pi^-}(s, a)$. However, in practice, the attacker usually only has access to the agent's current policy π, so it can also choose $a^- = \text{argmin} Q^\pi(s, a)$. Note that these two selections are different, as the agent's policy π is usually far away from the worst policy.

Lemma 26 (**Optimality of MaxWorst**). *Denote the set of optimal solutions to objective Equation* (II) *as* $H^{MaxWorst}$*, which include both versions of MaxWorst attacker formulations as we discussed above. Then there exist an MDP* $\mathcal{M}$ *and an agent policy* π*, such that* $H^{MaxWorst}$ *contains a non-optimal adversary* h^**, i.e.,* $H^{MaxWorst} \not\subset H^*_\epsilon$.

Proof of Lemma 26.

Case I: Using current policy to compute the target action

We prove this lemma by constructing the MDP in Figure 2.8 such that for any victim policy, there exists a reward configuration in which MaxWorst attacker is not optimal.

Here, let $r_1 = r(s_{11}|s_1, a_1), r_2 = r(s_{12}|s_1, a_2), r_3 = r(s_{21}|s_2, a_1), r_4 = r(s_{22}|s_2, a_2)$. Assuming

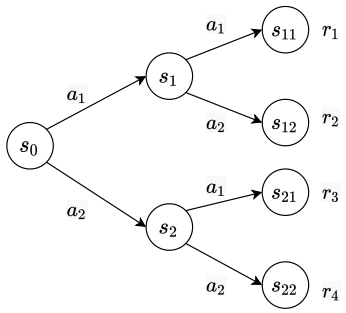


Figure 2.8: A simple MDP where the first version of MaxWorst Attacker cannot find the optimal adversary for a given victim policy.

all the other rewards are zero, transition dynamics are deterministic, and states $s_{11}, s_{12}, s_{21}, s_{22}$ are the terminal states. For the sake of simplicity, we also assume that the discount factor here $\gamma = 1$.

Now given a policy π such that $\pi(a_1|s_0) = \beta_0$, $\pi(a_1|s_1) = \beta_1$, and $\pi(a_2|s_2) = \beta_2$ ($\beta_0, \beta_1, \beta_2 \in [0, 1]$), consider the Adv-policy-set

$$\mathcal{B}_\epsilon^H(\pi) = \left\{ \pi' \in \Pi \;\middle|\; \|\pi'(\cdot|s_1) - \pi(\cdot|s_1)\| < \epsilon_0, \|\pi'(\cdot|s_1) - \pi(\cdot|s_1)\| < \epsilon_1, \|\pi'(\cdot|s_2) - \pi(\cdot|s_2)\| < \epsilon_2, \right\}.$$

We could find r_1, r_2, r_3, r_4 such that the following linear constraints hold:

$$\beta_1 r_1 + (1-\beta_1) r_2 > \beta_2 r_3 + (1-\beta_2) r_4 \Longleftrightarrow Q^\pi(s_0, a_1) > Q^\pi(s_0, a_2) \tag{2.27}$$

$$r_1 > r_2 \Longleftrightarrow Q^\pi(s_1, a_1) > Q^\pi(s_1, a_2) \tag{2.28}$$

$$r_3 > r_4 \Longleftrightarrow Q^\pi(s_2, a_1) > Q^\pi(s_2, a_2) \tag{2.29}$$

$$(\beta_1 - \epsilon_1) r_1 + (1 - \beta_1 + \epsilon_1) r_2 < (\beta_2 - \epsilon_2) r_3 + (1 - \beta_2 + \epsilon_2) r_4 \tag{2.30}$$

Now, given these constraints, the perturbed policy given by MaxWorst attaker satisfies $\pi_{h^{\text{MaxWorst}}}(a_1|s_0) = \beta_0 - \epsilon_0$, $\pi_{h^{\text{MaxWorst}}}(a_1|s_1) = \beta_1 - \epsilon_1$, and $\pi_{h^{\text{MaxWorst}}}(a_1|s_2) = \beta_2 - \epsilon_2$. However, consider another perturbed policy π_{h^*} in Adv-policy-set such that $\pi_{h^*}(a_1|s_0) = \beta_0 + \epsilon_0$, $\pi_{h^*}(a_1|s_1) = \beta_1 - \epsilon_1$, and $\pi_{h^*}(a_1|s_2) = \beta_2 - \epsilon_2$. We will prove that $V^{\pi_{h^*}}(s_1) < V^{\pi_{h^{\text{MaxWorst}}}}(s_1)$, and thus MaxWorst attacker is not optimal.

On the one hand,

$$V^{\pi_{h^{\text{MaxWorst}}}}(s_1) = (\beta_0 - \epsilon_0)\Big[(\beta_1 - \epsilon_1) r_1 + (1 - \beta_1 + \epsilon_1) r_2\Big] + (1 - \beta_0 + \epsilon_0)\Big[(\beta_2 - \epsilon_2) r_3 + (1 - \beta_2 + \epsilon_2) r_4\Big] \tag{2.31}$$

$$\begin{aligned} =&(\beta_0 - \epsilon_0)(\beta_1 - \epsilon_1) r_1 + (\beta_0 - \epsilon_0)(1 - \beta_1 + \epsilon_1) r_2 \\ &+ (1 - \beta_0 + \epsilon_0)(\beta_2 - \epsilon_2) r_3 + (1 - \beta_0 + \epsilon_0)(1 - \beta_2 + \epsilon_2) r_4 \end{aligned} \tag{2.32}$$

On the other hand,

$$V^{\pi_{h^*}}(s_1) = (\beta_0 + \epsilon_0)\Big[(\beta_1 - \epsilon_1) r_1 + (1 - \beta_1 + \epsilon_1) r_2\Big] + (1 - \beta_0 - \epsilon_0)\Big[(\beta_2 - \epsilon_2) r_3 + (1 - \beta_2 + \epsilon_2) r_4\Big] \tag{2.33}$$

$$\begin{aligned} =&(\beta_0 + \epsilon_0)(\beta_1 - \epsilon_1) r_1 + (\beta_0 + \epsilon_0)(1 - \beta_1 + \epsilon_1) r_2 \\ &+ (1 - \beta_0 - \epsilon_0)(\beta_2 - \epsilon_2) r_3 + (1 - \beta_0 - \epsilon_0)(1 - \beta_2 + \epsilon_2) r_4 \end{aligned} \tag{2.34}$$

Therefore,

$$V^{\pi_{h^*}}(s_1) - V^{\pi_{h^{\text{MaxWorst}}}}(s_1) = 2\epsilon_0(\beta_1 - \epsilon_1)r_1 + 2\epsilon_0(1 - \beta_1 + \epsilon_1)r_2$$

$$- 2\epsilon_0(\beta_2 - \epsilon_2)r_3 - 2\epsilon_0(1 - \beta_2 + \epsilon_2)r_4 \tag{2.35}$$

$$< 0 \text{ Because of the constraint Equation (2.30)} \tag{2.36}$$

Case II: Using worst policy to compute the target action

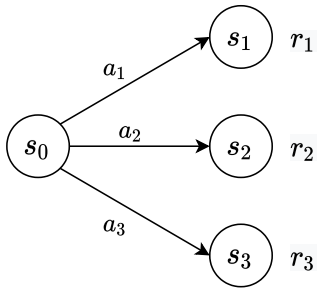


Figure 2.9: A simple MDP where the second version of MaxWorst Attacker cannot find the optimal adversary for a given victim policy.

Same as before, we construct a MDP where H^{MaxWorst} contains a non-optimal adversary. Let $r_1 = r(s_1|s_0, a_1), r_2 = r(s_2|s_0, a_2), r_3 = r(s_3|s_0, a_3)$. Assuming all the other rewards are zero, transition dynamics are deterministic, and states s_1, s_2, s_3 are the terminal states. For the sake of simplicity, we also assume that the discount factor here $\gamma = 1$.

Let pi be the given policy such that $\pi(a_1|s_0) = \beta_1$ and $\pi(a_2|s_0) = \beta_2$. Now without loss of generality, we assume $r_1 > r_2 > r_3 \quad (*)$. Then the worst policy π' satisfies that $\pi'(a_3|s_0) = 1$. Consider the Adv-policy-set $\mathcal{B}_\epsilon^H(\pi) = \Big\{\pi' \in \Pi \;\Big|\; \|\pi'(\cdot|s_0) - \pi(\cdot|s_0)\|_1 < \epsilon\Big\}$. Then $H^{\text{MaxWorst}} = \Big\{\pi' \in \Pi \;\Big|\; \pi'(a_3|s_0) = (1 - \beta_1 - \beta_2) + \epsilon\Big\}$.

Now consider two policies $\pi_{h^1}, \pi_{h^2} \in H^{\text{MaxWorst}}$, where $\pi_{h^1}(a_1|s_0) = \beta_1$, $\pi_{h^1}(a_2|s_0) = \beta_2 - \epsilon$, $\pi_{h^2}(a_1|s_0) = \beta_1 - \epsilon$, $\pi_{h^2}(a_2|s_0) = \beta_2$. Then $V^{\pi_{h^1}}(s_0) - V^{\pi_{h^2}}(s_0) = \epsilon(r_1 - r_2) > 0$. Therefore, $\pi_{h^1} \in H^{\text{MaxWorst}}$ but it's not optimal.

□

2.7.7.3 *TYPE III* - Minimize Q Value (MinQ).

Another idea of attacking [17, 19] is to craft perturbations such that the agent selects actions with minimized Q values at every step, i.e.,

$$h^{\text{MinQ}} \in \text{argmin}_{h \in H_\epsilon} \sum\nolimits_{a \in \mathcal{A}} \pi_h(a|s) \hat{Q}^\pi(s,a), \forall s \in \mathcal{S} \tag{III}$$

where $\hat{Q}$ is the approximated Q function of the agent's original policy. For example, Pattanaik et al.[17] directly use the agent's Q network (of policy π), while the Robust SARSA (RS) attack proposed by Zhang et al.[19] learns a more stable Q network for the agent's policy π. Note that in practice, this type of attack is usually applied to deterministic agents (e.g., DQN, DDPG, etc), then the objective becomes $\text{argmin}_{h \in H_\epsilon} \hat{Q}^\pi(s, \pi_h(s)), \forall s \in \mathcal{S}$ [17, 19, 39]. In this case, the MinQ attack is equivalent to the MaxWorst attack with the current policy as the target.

Lemma 27 (Optimality of MinQ). *Denote the set of optimal solutions to objective Equation (III) as H^{MinQ}, which include both versions of MinQ attacker formulations as we discussed above. Then there exist an MDP $\mathcal{M}$ and an agent policy π, such that H^{MinQ} contains a non-optimal adversary h^*, i.e., $H^{MinQ} \not\subset H^*_\epsilon$.*

Proof of Lemma 26.

Case I: For a deterministic victim

In the deterministic case

$$h^{\text{MinQ}} \in \text{argmin}_{h \in H_\epsilon} \hat{Q}^\pi(s, \pi_h(s)) = \text{argmax}_{h \in H_\epsilon} \pi_h(\text{argmin}_a \hat{Q}^\pi(s,a)|s), \forall s \in \mathcal{S} \tag{III$_D$}$$

In this case, the objective is equivalent to objective Equation (II), thus Lemma 27 holds.

Case II: For a stochastic victim

In this case, we consider the MDP in Figure 2.8 and condition Equation (2.27) to Equation (2.30). Then the MinQ objective gives $\pi_{h^{\text{MinQ}}}(a_1|s_0) = \beta_0 - \epsilon_0$, $\pi_{h^{\text{MinQ}}}(a_1|s_1) = \beta_1 - \epsilon_1$, and $\pi_{h^{\text{MinQ}}}(a_1|s_2) = \beta_2 - \epsilon_2$.

According to the proof of the first case of Lemma 26, $\pi_{h^{\text{MinQ}}} = \pi_{h^{\text{MaxWorst}}}$ is not an optimal adversary. Thus Lemma 27 holds.

□

2.7.7.4 *TYPE IV* - Maximize Difference (MaxDiff).

The MAD attack proposed by Zhang et al. [19] is to maximize the distance between the perturbed policy π_h and the clean policy π, i.e.,

$$h^{\text{MaxDiff}} \in \text{argmax}_{h \in H_\epsilon} \text{D}_{\text{TV}}[\pi_h(\cdot|s)||\pi(\cdot|s)], \forall s \in \mathcal{S} \tag{IV}$$

where TV denotes the total variance distance between two distributions. In practical implementations, the TV distance can be replaced by the KL-divergence, as $\text{D}_{\text{TV}}[\pi_h(\cdot|s)||\pi(\cdot|s)] \leq (\text{D}_{\text{KL}}[\pi_h(\cdot|s)||\pi(\cdot|s)])^2$. This type of attack is inspired by the fact that if two policies select actions with similar action distributions on all the states, then the value of the two policies is also small (see Theorem 5 in [19]).

Lemma 28 (Optimality of MaxDiff). *Denote the set of optimal solutions to objective Equation (IV) as $H^{MaxDiff}$. There exist an MDP $\mathcal{M}$ and an agent policy π, such that $H^{MaxDiff}$ contains a non-optimal adversary h^*, i.e., $H^{MaxDiff} \not\subset H^*_\epsilon$.*

Proof of Lemma 28. The proof follows from the proof of lemma 25. In the MDP we constructed, $\pi' = \beta_1 - \epsilon_1, \pi_{h^{\text{MinBest}}}(a_1|s_2) = \beta_2 - \epsilon_2$ is one of the policies that has the maximum KL divergence from the victim policy within Adv-policy-set. However, as we proved in 25, this is not the

optimally perturbed policy. Therefore, MaxDiff attacker may not be optimal.

□

2.7.8 Additional Experiment Details and Results

2.7.8.1 Implementation Details

A. Atari Experiments

In this section we report the configurations and hyperparameters we use for DQN, A2C and ACKTR in Atari environments. We use *GeForce RTX 2080 Ti* GPUs for all the experiments.

DQN Victim. We compare PA-AD algorithm with other attacking algorithms on 7 Atari games. For DQN, we take the softmax of the Q values $Q(s, \cdot)$ as the victim policy $\pi(\cdot|s)$ as in prior works [4]. For these environments, we use the wrappers provided by stable-baselines [44], where we clip the environment rewards to be -1 and 1 during training and stack the last 4 frames as the input observation to the DQN agent. For the victim agent, we implement Double Q learning [45] and prioritized experience replay [46]. The clean DQN agents are trained for 6 million frames, with a learning rate 0.00001 and the same network architecture and hyperparameters as the ones used in [1]. In addition, we use a replay buffer of size 5×10^5. Prioritized replay buffer sampling is used with $\alpha = 0.6$ and β increases from 0.4 to 1 linearly during training. During evaluation, we execute the agent's policy without epsilon greedy exploration for 1000 episodes.

A2C Victim. For the A2C victim agent, we also use the same preprocessing techniques and convolutional layers as the one used in [1]. Besides, values and policy network share the same CNN layers and a fully-connected layer with 512 hidden units. The output layer is a categorical distribution over the discrete action space. We use 0.0007 as the initial learning rate and apply

linear learning rate decay, and we train the victim A2C agent for 10 million frames. During evaluation, the A2C victim executes a stochastic policy (for every state, the action is sampled from the categorical distribution generated by the policy network). Our implementation of A2C is mostly based on an open-source implementation by Kostrikov [47].

ACKTR Adversary. To train the director of PA-AD and the adversary in SA-RL, we use ACKTR [42] with the same network architecture as A2C. We train the adversaries of PA-AD and SA-RL for the same number of steps for a fair comparison. For the DQN victim, we use a learning rate 0.0001 and train the adversaries for 5 million frames. For the A2C victim, we use a learning rate 0.0007 and train the adversaries for 10 million frames. Our implementation of ACKTR is mostly based on an open-source implementation by Kostrikov [47].

Heuristic Attackers. For the MinBest attacker, we following the algorithm proposed by [4] which uses FGSM to compute adversarial state perturbations. The MinBest + Momentum attacker is implemented according to the algorithm proposed by [21], and we set the number of iterations to be 10, the decaying factor μ to be 0.5 (we tested $0.01, 0.1, 0.5, 0.9$ and found 0.5 is relatively better while the difference is minor). Our implementation of the MinQ attacker follows the gradient-based attack by [17], and we also set the number of iterations to be 10. For the MaxDiff attacker, we refer to Algorithm 3 in [19] with the number of iterations equal to 10. In addition, we implement a random attacker which perturbs state s to $\tilde{s} = s + \epsilon \text{sign}(\mu)$, where μ is sampled from a standard multivariate Gaussian distribution with the same dimension as s.

B. MuJoCo Experiments

For four OpenAI Gym MuJoCo continuous control environments, we use PPO with the original fully connected (MLP) structure as the policy network to train the victim policy. For robustness evaluations, the victim and adversary are both trained using PPO with independent

value and policy optimizers. We complete all the experiments on MuJoCo using *32GB Tesla V100*.

PPO Victim We directly use the well-trained victim model provided by [19].

PPO Adversary Our PA-AD adversary is trained by PPO and we use a grid search of a part of adversary hyperparameters (including learning rates of the adversary policy network and policy network, the entropy regularization parameter and the ratio clip ϵ for PPO) to train the adversary as powerful as possible. The reported optimal attack result is from the strongest adversary among all 50 trained adversaries.

Other Attackers For Robust Sarsa (RS) attack, we use the implementation and the optimal RS hyperparameters from [19] to train the robust value function to attack the victim. The reported RS attack performance is the best one over the 30 trained robust value functions. For MaxDiff attack, the maximal action difference attacker is implemented referring to [19]. For SA-RL attacker, following [18], the hyperparameters is the same as the optimal hyperparameters of vanilla PPO from a grid search. And the training steps are set for different environments. For the strength of SA-PPO regularization κ, we choose from 1×10^{-6} to 1 and report the worst-case reward.

Robust Training For ATLA [18], the hyperparameters for both victim policy and adversary remain the same as those in vanilla PPO training. To ensure sufficient exploration, we run a small-scale grid search for the entropy bonus coefficient for agent and adversary. The experiment results show that a larger entropy bonus coefficient allows the agent to learn a better policy for the continual-improving adversary. In robust training experiments, we use larger training steps in all the MuJoCo environments to guarantee policy convergence. We train 5 million steps in Hopper, Walker, and HalfCheetah environments and 10 million steps for Ant. For reproducibility, the final

results we reported are the experimental performance of the agent with medium robustness from 21 agents training with the same hyperparameter set.

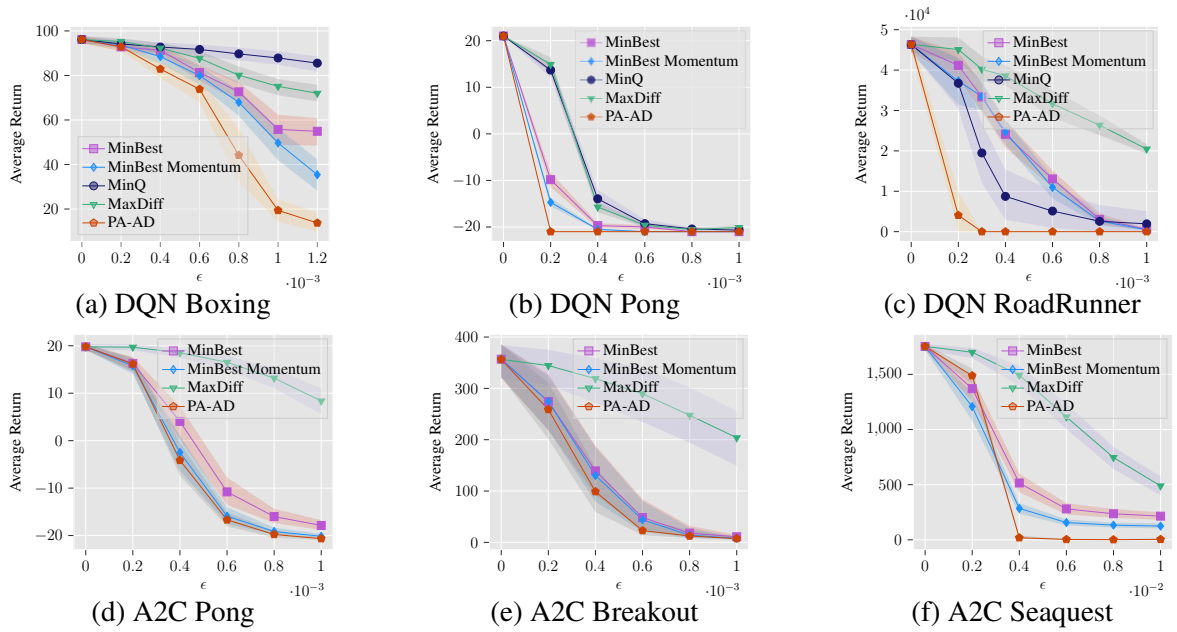
2.7.8.2 Attacking Performance with Various Budgets

In Table 2.1, we report the performance of our PA-AD attacker under a chosen epsilon across different environments. To see how PA-AD algorithm performs across different values of ϵ's, here we select three Atari environments each for DQN and A2C victim agents and plot the performance of PA-AD under various ϵ's compared with the baseline attackers in Figure 2.10. We can see from the figures that our PA-AD universally outperforms baseline attackers concerning various ϵ's.

In Table 2.2, we provide the evaluation results of PA-AD under a commonly unused epsilon in four MuJoCo experiments ([18, 19]) to show that PA-AD attacker also has the best attacking performance compared with other attackers under different ϵ's in Figure 2.11.

2.7.8.3 Hyperparameter Test

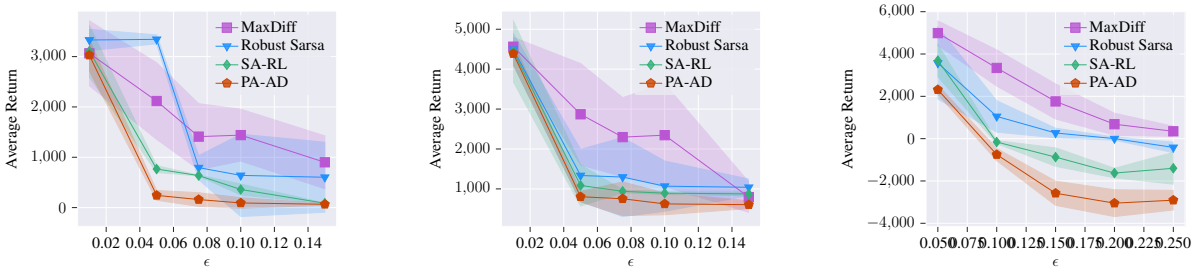
In our Actor-Director Framework, solving an optimal actor is a constraint optimization problem. Thus, in our algorithm, we instead use Lagrangian relaxation for the actor's constraint optimization. In this section, we report the effects of different choices of the relaxation hyperparameter λ on the final performance of our algorithm. Although we set λ by default to be 1 and keep it fixed throughout all of the other experiments, here we find that in fact, difference choice of λ has only minor impact on the performance of the attacker. This result demonstrates that our PA-AD algorithm is robust to different choices of relaxation hyperparameters.



(a) DQN Boxing (b) DQN Pong (c) DQN RoadRunner

(d) A2C Pong (e) A2C Breakout (f) A2C Seaquest

Figure 2.10: Comparison of different attack methods against DQN and A2C victims in Atari w.r.t. different budget ϵ's.



(a) PPO Hopper (b) PPO Walker2d (c) PPO Ant

Figure 2.11: Comparison of different attack methods against PPO victims in MuJoCo w.r.t. different budget ϵ's.

Table 2.4: Performance of PA-AD across difference choices of the relaxation hyperparameter λ

	Pong	Boxing
Nature Reward	21 ± 0	96 ± 4
$\lambda = 0.2$	-19 ± 2	16 ± 12
$\lambda = 0.4$	-18 ± 2	17 ± 12
$\lambda = 0.6$	-20 ± 2	19 ± 15
$\lambda = 0.8$	-19 ± 2	14 ± 12
$\lambda = 1.0$	-19 ± 2	15 ± 12
$\lambda = 2.0$	-20 ± 1	21 ± 15
$\lambda = 5.0$	-20 ± 1	19 ± 14

(a) Atari

	Ant	Walker
Nature Reward	5687 ± 758	4472 ± 635
$\lambda = 0.2$	-2274 ± 632	897 ± 157
$\lambda = 0.4$	-2239 ± 716	923 ± 132
$\lambda = 0.6$	-2456 ± 853	954 ± 105
$\lambda = 0.8$	-2597 ± 662	872 ± 162
$\lambda = 1.0$	-2580 ± 872	804 ± 130
$\lambda = 2.0$	-2378 ± 794	795 ± 124
$\lambda = 5.0$	-2425 ± 765	814 ± 140

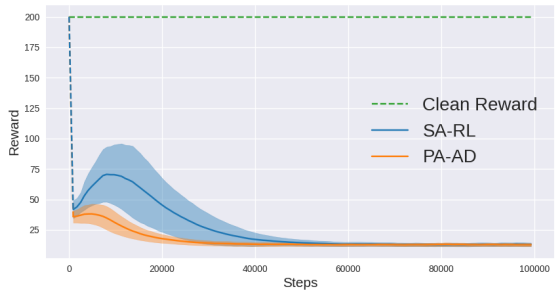
(b) Mujoco

2.7.8.4 Empirical Comparison between PA-AD and SA-RL

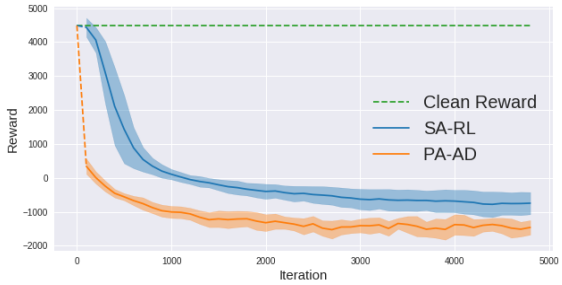
In this section, we provide more empirical comparison between PA-AD and SA-RL. Note that PA-AD and SA-RL are different in terms of their applicable scenarios: SA-RL is a black-box attack methods, while PA-AD is a white-box attack method. When the victim model is known, we can see that by a proper exploitation of the victim model, PA-AD demonstrates better attack performance, higher sample and computational efficiency, as well as higher scalability.

PA-AD has better convergence property than SA-RL. In Figure 2.12, we plot the learning curves of SA-RL and PA-AD in the CartPole environment and the Ant environment. Compared with SA-RL attacker, PA-AD has a higher attacking strength in the beginning and converges much faster. In Figure 2.12b, we can see that PA-AD has a "warm-start" (the initial reward of the victim is already significantly reduced) compared with SA-RL attacker which starts from scratch. This is because PA-AD always tries to maximize the distance between the perturbed policy and the original victim policy in every step according to the actor function (G). So in the beginning of learning, PA-AD works similarly to the MaxDiff attacker, while SA-RL works similarly to a

random attacker. We also note that although PA-AD algorithm is proposed particularly for environments that have state spaces much larger than action spaces, in CartPole where the state dimensions is fewer than the number of actions, PA-AD still works better than SA-RL because of the distance maximization.



(a) Learning curve of SA-RL and PA-AD attacker against an A2C victim in CartPole.



(b) Learning curve of SA-RL and PA-AD attacker against a PPO victim in Ant.

Figure 2.12: Comparison of convergence rate between SA-RL and PA-AD in Ant and Cartpole. Results are averaged over 10 random seeds.

PA-AD is more computationally efficient than SA-RL. Our experiments in Section 2.5 show that PA-AD converges to a better adversary than SA-RL given the same number of training steps, which verifies the sample efficiency of PA-AD. Another aspect of efficiency is based on the computational resources, including running time and required memory. For RL algorithms, the computation cost comes from the interaction with the environment (the same for SA-RL and PA-AD) and the policy/value update. If the state space $\mathcal{S}$ is higher-dimensional than the action space $\mathcal{A}$, then SA-RL requires a larger policy network than PA-AD since SA-RL has a higher-dimensional output, and thus SA-RL has more network parameters than PA-AD, which require more memory cost and more computation operations. On the other hand, PA-AD requires to solve an additional optimization problem defined by the actor objective (G) or (G_D). In our implementation, we use FGSM which only requires one-step gradient computation and is thus efficient. But if more advanced optimization algorithms (e.g. PGD) are used, more computations

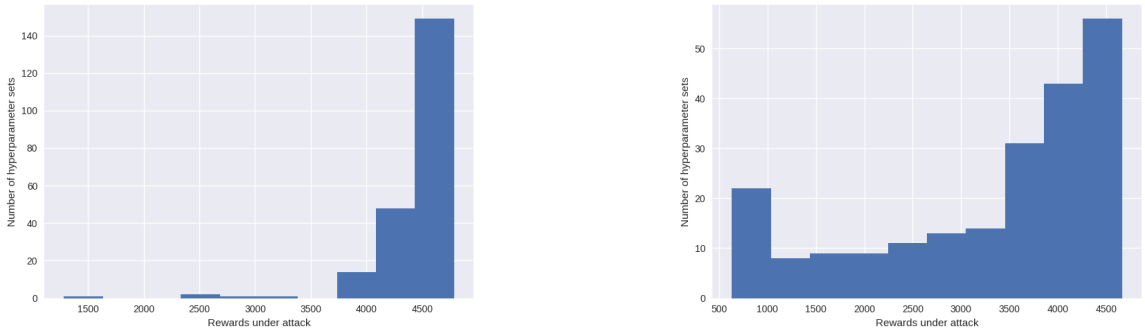
may be needed. In summary, if $\mathcal{S}$ is much larger than $\mathcal{A}$, PA-AD is more computational efficient than SA-RL; if $\mathcal{A}$ is much larger than $\mathcal{S}$, SA-RL is more efficient than PA-AD; if the sizes of $\mathcal{S}$ and $\mathcal{A}$ are similar, PA-AD may be slightly more expensive than SA-RL, depending on the optimization methods selected for the actor.

To verify the above analysis, we compare computational training time for training SA-RL and PA-AD attackers, which shows that PA-AD is more computationally efficient. Especially on the environment with high-dimensional states like Ant, PA-AD takes significantly less training time than SA-RL (and finds a better adversary than SA-RL), which quantifies the efficiency of our algorithm in empirical experiments.

Method	**Hopper**	**Walker2d**	**HalfCheetah**	**Ant**
SA-RL	1.80	1.92	1.76	4.88
PA-AD	1.43	1.46	1.40	3.76

Table 2.5: Average training time (in hours) of SA-RL and PA-AD in MuJoCo environments, using GeForce RTX 2080 Ti GPUs. For Hopper, Walker2d and HalfCheetah, SA-RL and PA-AD are both trained for 2 million steps; for Ant, SA-RL and PA-AD are both trained for 5 million steps

PA-AD is less sensitive to hyperparameters settings than SA-RL. In addition to better final attacking results and convergence property, we also observe that PA-AD is much less sensitive to hyerparameter settings compared to SA-RL. On the Walker environment, we run a grid search over 216 different configurations of hyperparameters, including actor learning rate, critic learning rate, entropy regularization coefficient, and clipping threshold in PPO. Here for comparison we plot two histograms of the agent's final attacked results across different hyperparameter configurations.



(a) SA-RL Attacker (b) PA-AD Attacker

Figure 2.13: Histograms of victim rewards under different hyperparameter settings of SA-RL and PA-AD on Walker.

The perturbation radius is set to be 0.05, for which the mean reward reported by [19] is 1086. However, as we can see from this histogram, only one out of the 216 configurations of SA-RL achieves an attacking reward within the range 1000-2000, while in most hyperparameter settings, the mean attacked return lies in the range 4000-4500. In contrast, about 10% hyperparameter settings of PA-AD algorithm are able to reduce the reward to 500-1000, and another 10% settings could reduce the reward to 1000-2000. Therefore, the performance of PA-AD attacker is generally better and more robust across different hyperparameter configurations than SA-RL.

2.7.8.5 Robust Training Efficiency on MuJoCo by PA-ATLA

In the ATLA process proposed by [18], one alternately trains an agent and an adversary. As a result, the agent policy may learn to adapt to the specific type of attacker it encounters during training. In Table 2.3, we present the performance of our robust training method PA-ATLA-PPO compared with ATLA-PPO under different types of attacks during testing. ATLA-PPO uses SA-RL to train the adversary, while PA-ATLA-PPO uses PA-AD to train the adversary during alternating training. As a result, we can see that ATLA-PPO models perform better under the SA-

RL attack, and PA-ATLA-PPO performs better under the PA-AD attack. However, the advantage of ATLA-PPO over PA-ATLA-PPO against SA-RL attack is much smaller than the advantage of PA-ATLA-PPO over ATLA-PPO against PA-AD attack. In addition, our PA-ATLA-PPO models significantly outperform ATLA-PPO models against other heuristic attack methods, and achieve higher average rewards across all attack methods. Therefore, PA-ATLA-PPO is generally more robust than ATLA-PPO.

Furthermore, the efficiency of training an adversary could be the bottleneck in the ATLA [18] process for practical usage. Section 2.7.8.4 suggests that our PA-AD generally converges faster than SA-RL. Therefore, when the computation resources are limited, PA-ATLA-PPO can train robust agents faster than ATLA-PPO. We conduct experiments on continuous control environments to empirically show the efficiency comparison between PA-ATLA-PPO and ATLA-PPO. In Table 2.6, we show the robustness performance of two ATLA methods with 2 million training steps for Hopper, Walker and Halfcheetah and 5 million steps for Ant (Compared with results in Table 2.3, we have reduced training steps by half or more). It can be seen that our PA-ATLA-PPO models still significantly outperform the original ATLA-PPO models under different types of attacks. More importantly, our PA-ATLA-PPO achieves higher robustness under SA-RL attacks in Walker and Ant, suggesting the efficiency and effectiveness of our method.

2.7.8.6 Attacking Robustly Trained Agents on Atari

In this section, we show the attack performance of our proposed algorithm PA-AD against DRL agents that are trained to be robust by prior works [19, 39] in Atari games.

Zhang et al. [19] propose **SA-DQN**, which minimizes the action change under possible

Environment	ϵ	step(million)	Model	Natural Reward	RS [19]	SA-RL [18]	PA-AD (ours)	Average reward across attacks
Hopper	0.075	2	ATLA-PPO	1763 ± 818	1349 ± 174	1172 ± 344	$\mathbf{477 \pm 30}$	999.3
			PA-ATLA-PPO	2164 ± 121	1720 ± 490	1119 ± 123	$\mathbf{1024 \pm 188}$	1287.7
Walker	0.05	2	ATLA-PPO	3183 ± 842	2405 ± 529	2170 ± 1032	$\mathbf{516 \pm 47}$	1697.0
			PA-ATLA-PPO	3206 ± 445	2749 ± 106	2332 ± 198	$\mathbf{1072 \pm 247}$	2051.0
Halfcheetah	0.15	2	ATLA-PPO	4871 ± 112	3781 ± 645	3493 ± 372	$\mathbf{856 \pm 118}$	2710.0
			PA-ATLA-PPO	5257 ± 94	4012 ± 290	3329 ± 183	$\mathbf{1670 \pm 149}$	3003.7
Ant	0.15	5	ATLA-PPO	3267 ± 51	3062 ± 149	2208 ± 56	$\mathbf{-18 \pm 100}$	1750.7
			PA-ATLA-PPO	3991 ± 71	3364 ± 254	2685 ± 41	$\mathbf{2403 \pm 82}$	2817.3

Table 2.6: Average episode rewards $\pm$ standard deviation of robust models with fewer training steps under different evasion attack methods. Results are averaged over 50 episodes. We bold the strongest attack in each row. The gray cells are the most robust agents with the highest average rewards across all attacks.

state perturbations within ℓ_p norm ball, i.e., to minimize the extra loss

$$\mathcal{R}_{\text{DQN}}(\theta) := \sum_s \max \left\{ \max_{\hat{s} \in B(s)} \max_{a \neq a^*} Q_\theta(\hat{s}, a) - Q_\theta\left(\hat{s}, a^*(s)\right), -c \right\} \tag{2.37}$$

where θ refers to the Q network parameters, $a^*(s) = \text{argmax}_a Q_\theta(a|s)$, and c is a small constant. Zhang et al. [19] solve the above optimization problem by a convex relaxation of the Q network, which achieves 100% action certification (i.e. the rate that action changes with a constrained state perturbation) in Pong and Freeway, over 98% certification in BankHeist and over 47% certification in RoadRunner under attack budget $\epsilon = 1/255$.

Oikarinen et al. [39] propose another robust training method named RADIAL-RL. By adding a adversarial loss to the classical loss of the RL agents, and solving the adversarial loss with interval bound propagation, the proposed **RADIAL-DQN** and **RADIAL-A3C** achieve high rewards in Pong, Freeway, BankHeist and RoadRunner under attack budget $\epsilon = 1/255$ and $\epsilon = 3/255$.

Implementation of the Robust Agents and Environments. We directly use the trained SA-DQN agents provided by [19], as well as RADIAL-DQN and RADIAL-A3C agents pro-

vided by [39]. During test time, the agents take actions deterministically. In order to reproduce the results in these papers, we use the same environment configurations as in [19] and [39], respectively. But note that the environment configurations of SA-DQN and RADIAL-RL are simpler versions of the traditional Atari configurations we use (described in Section 2.7.8.1). Both SA-DQN and RADIAL-RL use a single frame instead of the stacking as 4 frames. Moreover, SA-DQN restricts the number of actions as 6 (4 for Pong) in each environment, although the original environments have 18 actions (6 for Pong). The above simplifications in environments can make robust training easier since the dimensionality of the input space is much smaller, and the number of possible outputs is restricted.

Attack Methods In experiments, we find that the robust agents are much harder to attack than vanilla agents in Atari games, as claimed by the robust training papers [19, 39]. A reason is that Atari games have discrete action spaces, and leading an agent to make a different decision at a state with a limited perturbation could be difficult. Therefore, we use a 30-step Projected Gradient Descent for all attack methods (with step size $\epsilon/10$), including MinBest [4] and our PA-AD which use FGSM for attacking vanilla models. Note that the PGD attacks used by [19] and [39] in their experiments are the same as the MinBest-PGD attack we use. For our PA-AD, we use PPO to train the adversary since PPO is relatively stable. The learning rate is set to be $5\mathrm{e}-4$, and the clip threshold is 0.1. Note that SA-DQN, RADIAL-DQN and RADIAL-A3C agents all take deterministic actions, so we use the deterministic formulation of PA-AD as described in Section 2.7.2.1. In our implementation, we simply use a CrossEntropy loss for the actor as in Equation (2.38).

$$g_D(\widehat{a}, s) = \mathrm{argmin}_{s' \in B_\epsilon(s)} \mathsf{CrossEntropy}(\pi(s'), \widehat{a}). \tag{2.38}$$

	Environment	Natural Reward	ϵ	Random	MinBest [4]	MinBest + Momentum [21]	MinQ [17]	MaxDiff [19]	PA-AD (ours)
SA-DQN	RoadRunner	46440 ± 5797	$\frac{1}{255}$	45032 ± 7125	40422 ± 8301	43856 ± 5445	42790 ± 8456	45946 ± 8499	$\mathbf{38652 \pm 6550}$
	BankHeist	1237 ± 11	$\frac{1}{255}$	1236 ± 12	1235 ± 15	1233 ± 17	1237 ± 14	1236 ± 13	1237 ± 14
RADIAL -DQN	RoadRunner	39102 ± 13727	$\frac{1}{255}$	41584 ± 8351	41824 ± 7858	42330 ± 8925	40572 ± 9988	42014 ± 8337	$\mathbf{38214 \pm 9119}$
			$\frac{3}{255}$	23766 ± 6129	9808 ± 4345	35598 ± 8191	39866 ± 6001	18994 ± 6451	$\mathbf{1366 \pm 3354}$
	BankHeist	1060 ± 95	$\frac{1}{255}$	1037 ± 103	991 ± 105	$\mathbf{988 \pm 102}$	1021 ± 96	1042 ± 112	999 ± 100
			$\frac{3}{255}$	1011 ± 130	801 ± 114	460 ± 310	842 ± 33	1023 ± 110	$\mathbf{397 \pm 172}$
RADIAL -A3C	RoadRunner	30854 ± 7281	$\frac{1}{255}$	30828 ± 7297	31296 ± 7095	31132 ± 6861	30838 ± 5743	32038 ± 6898	$\mathbf{30550 \pm 7182}$
			$\frac{3}{255}$	30690 ± 7006	30198 ± 6075	29936 ± 5388	29988 ± 6340	31170 ± 7453	$\mathbf{29768 \pm 5892}$
	BankHeist	847 ± 31	$\frac{1}{255}$	847 ± 31	847 ± 33	848 ± 31	848 ± 31	848 ± 31	848 ± 31
			$\frac{3}{255}$	848 ± 31	644 ± 158	822 ± 11	842 ± 33	834 ± 30	$\mathbf{620 \pm 168}$

Table 2.7: Average episode rewards $\pm$ standard deviation of SA-DQN, RADIAL-DQN, RADIAL-A3C robust agents under different evasion attack methods in Atari environments RoadRunner and BankHeist. All attack methods use 30-step PGD to compute adversarial state perturbations. Results are averaged over 50 episodes. In each row, we bold the strongest attack, except for the rows where none of the attacker reduces the reward significantly (which suggests that the corresponding agent is relatively robust).

Experiment Results In Table 2.7, we reproduce the results reported by [19] and [39], and demonstrate the average rewards gained by these robust agents under different attacks in RoadRunner and BankHeist. Note that SA-DQN is claimed to be robust to attacks with budget $\epsilon = 1/255$, and RADIAL-DQN and RADIAL-A3C are claimed to be relatively robust against up to $\epsilon = 3/255$ attacks. (ℓ_∞ is used in both papers.) So we use the same ϵ's for these agents in our experiments.

It can be seen that compared with vanilla agents in Table 2.1, SA-DQN, RADIAL-DQN and RADIAL-A3C are more robust due to the robust training processes. However, in some environments, PA-AD can still decrease the rewards of the agent significantly. For example, in RoadRunner with $\epsilon = 3/255$, RADIAL-DQN gets 1k+ reward against our PA-AD attack, although RADIAL-DQN under other attacks can get 10k+ reward as reported by [39]. In contrast, we find that RADIAL-A3C is relatively robust, although the natural rewards gained by RADIAL-

A3C are not as high as RADIAL-DQN and SA-DQN. Also, as SA-DQN achieves over 98% action certification in BankHeist, none of the attackers is able to noticeably reduce its reward with $\epsilon = 1/255$.

Therefore, our PA-AD can approximately evaluate the worst-case performance of an RL agent under attacks with fixed constraints, i.e., *PA-AD can serve as a "detector" for the robustness of RL agents.* For agents that perform well under other attacks, PA-AD may still find flaws in the models and decrease their rewards; for agents that achieve high performance under PA-AD attack, they are very likely to be robust against other attack methods.

2.7.8.7 Improving Robustness on Atari by PA-ATLA

Note that different from SA-DQN [19] and RADIAL-RL [39] discussed in Section 2.7.8.6, we use the traditional Atari configurations [1] without any simplification (e.g. disabling frame stacking, or restricting action numbers). We aim to improve the robustness of the agents in original Atari environments, as in real-world applications, the environments could be complex and unchangeable.

Baselines. We propose PA-ATLA-A2C by combining our PA-AD and the ATLA framework proposed by [18]. We implement baselines including vanilla A2C, adversarially trained A2C (with MinBest [4] and MaxDiff [19] adversaries attacking 50 frames). SA-A2C [19] is implemented using SGLD and convex relaxations in Atari environments.

In Table 6, naive adversarial training methods have unreliable performance under most strong attacks and SA-A2C is ineffective under PA-AD strongest attack. To provide evaluation using different ϵ, we provide the attack rewards of all robust models with different attack budgets

ϵ. Under all attacks with different ϵ value, PA-ATLA-A2C models outperform all other robust models and achieve consistently better average rewards across attacks. We can observe that our PA-ATLA-A2C training method can considerably enhance the robustness in Atari environments.

Model	Natural Reward	ϵ	Random	MinBest [4]	MaxDiff [19]	SA-RL [18]	PA-AD (ours)	Average reward across attacks
A2C vanilla	1228 ± 93	1/255	1223 ± 77	972 ± 99	1095 ± 107	1132 ± 30	$\mathbf{436 \pm 74}$	971.6
		3/255	1064 ± 129	697 ± 153	913 ± 164	928 ± 124	$\mathbf{284 \pm 116}$	777.2
A2C (adv: MinBest [4])	948 ± 94	1/255	932 ± 69	927 ± 30	936 ± 11	940 ± 103	$\mathbf{704 \pm 19}$	887.8
		3/255	874 ± 51	813 ± 32	829 ± 27	843 ± 126	$\mathbf{521 \pm 72}$	774.2
A2C (adv: MaxDiff [19])	743 ± 29	1/255	756 ± 42	702 ± 89	752 ± 79	749 ± 85	$\mathbf{529 \pm 45}$	697.6
		3/255	712 ± 109	638 ± 133	694 ± 115	686 ± 110	$\mathbf{403 \pm 101}$	626.6
SA-A2C[18]	1029 ± 152	1/255	1054 ± 31	902 ± 89	1070 ± 42	1067 ± 18	$\mathbf{836 \pm 70}$	985.8
		3/255	985 ± 47	786 ± 52	923 ± 52	972 ± 126	$\mathbf{644 \pm 153}$	862.0
PA-ATLA-A2C (ours)	1076 ± 56	1/255	1055 ± 204	957 ± 78	1069 ± 94	1045 ± 143	$\mathbf{862 \pm 106}$	997.6
		3/255	1026 ± 78	842 ± 154	967 ± 82	976 ± 159	$\mathbf{757 \pm 132}$	913.6

Table 2.8: Average episode rewards $\pm$ standard deviation over 50 episodes of A2C, A2C with adv. training, SA-A2C and our PA-ATLA-A2C robust models under different evasion attack methods in Atari environment BankHeist. In each row, we bold the strongest attack. The gray cells are the most robust agents with the highest average rewards across all attacks.

Chapter 3: A Generic Training Framework for Improving the Robustness of RL

3.1 Introduction

Deep reinforcement learning (DRL) has achieved impressive results by using deep neural networks (DNN) to learn complex policies in large-scale tasks. However, well-trained DNNs may drastically fail under adversarial perturbations of the input [48, 49]. Therefore, before deploying DRL policies to real-life applications, it is crucial to improve the robustness of deep policies against adversarial attacks, especially worst-case attacks that maximally depraves the performance of trained agents [8].

A line of regularization-based robust methods [19, 39, 50] focuses on improving the robustness of the DNN itself and regularizes the policy network to output similar actions under bounded state perturbations. However, different from supervised learning problems, the vulnerability of a deep policy comes not only from the DNN approximator, but also from the dynamics of the RL environment [18]. These regularization-based methods neglect the intrinsic vulnerability of policies under the environment dynamics, and thus may still fail under strong attacks [8].

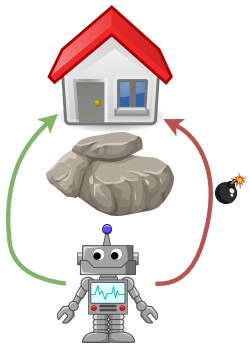


Figure 3.1: Policies have different vulnerabilities.

For example, in the go-home task shown in Figure 3.1, both the green policy and the red policy arrive home without rock collision, when there is no attack. However, although regularization-

based methods may ensure a minor action change under a state perturbation, the red policy may still be susceptible to a low reward under attacks, as a very small divergence can lead it to the bomb. On the contrary, the green policy is more robust to adversarial attacks since it stays away from the bomb. Therefore, besides promoting the robustness of DNN approximators (such as the policy network), it is also important to learn a policy with stronger intrinsic robustness.

There is another line of work considering the long-term robustness of a deep policy under strong adversarial attacks. In particular, it is theoretically proved [8, 19] that the strongest (worst-case) attacker against a policy can be learned as an RL problem, and training the agent under such a learned attacker can result in a robust policy. Zhang et al. [18] propose the *Alternating Training with Learned Adversaries (ATLA)* framework, which alternately trains an RL agent and an RL attacker. Sun et al. [8] further propose PA-ATLA, which alternately trains an agent and the proposed more efficient PA-AD RL attacker, obtaining state-of-the-art robustness in many MuJoCo environments. However, training an RL attacker requires extra samples from the environment, and the attacker's RL problem may even be more difficult and sample expensive to solve than the agent's original RL problem [8, 18], especially in large-scale environments such as Atari games with pixel observations. Therefore, although ATLA and PA-ATLA are able to achieve high long-term reward under attacks, they double the computational burden and sample complexity to train the robust agent.

The above analysis of existing literature suggests two main challenges in improving the adversarial robustness of DRL agents: (1) correctly characterizing the long-term reward vulnerability of an RL policy, and (2) efficiently training a robust agent without requiring much more effort than vanilla training. To tackle these challenges, in this chapter, we propose a generic and efficient robust training framework named *Worst-case-aware Robust RL (WocaR-RL)* that

estimates and improves the long-term robustness of an RL agent.

WocaR-RL has 3 key mechanisms. First, WocaR-RL introduces a novel *worst-attack Bellman operator* which uses existing off-policy samples to estimate the lower bound of the policy value under the worst-case attack. Compared to prior works [8, 18] which attempt to learn the worst-case attack by RL methods, WocaR-RL does not require any extra interaction with the environment. Second, using the estimated worst-case policy value, WocaR-RL optimizes the policy to select actions that not only achieve high natural future reward, but also achieve high worst-case reward when there are adversarial attacks. Therefore, WocaR-RL learns a policy with less intrinsic vulnerability. Third, WocaR-RL regularizes the policy network with a carefully designed state importance weight. As a result, the DNN approximator tolerates state perturbations, especially for more important states where decisions are crucial for future reward. The above 3 mechanisms can also be interpreted from a geometric perspective of adversarial policy learning, as detailed in Section 3.4.1.

Summary of Contributions

(1) We provide an approach to estimate the worst-case value of any policy under any bounded ℓ_p adversarial attacks. This helps evaluate the robustness of a policy without learning an attacker which requires extra samples and exploration.

(2) We propose a novel and principled robust training framework for RL, named *Worst-case-aware Robust RL (WocaR-RL)*, which characterizes and improves the worst-case robustness of an agent. WocaR-RL can be used to robustify existing DRL algorithms (e.g. PPO [16], DQN [15]).

(3) We show by experiments that WocaR-RL achieve **improved robustness** against various adversarial attacks as well as **higher efficiency**, compared with state-of-the-art (SOTA) robust RL methods in many MuJoCo and Atari games. For example, compared to the SOTA algorithm PA-

ATLA-PPO [8] in the Walker environment, we obtain 20% more worst-case reward (under the strongest attack algorithm), with only about 50% training samples and 50% running time. Moreover, WocaR-RL learns **more interpretable "robust behaviors"** than PA-ATLA-PPO in Walker as shown in Figure 3.2.

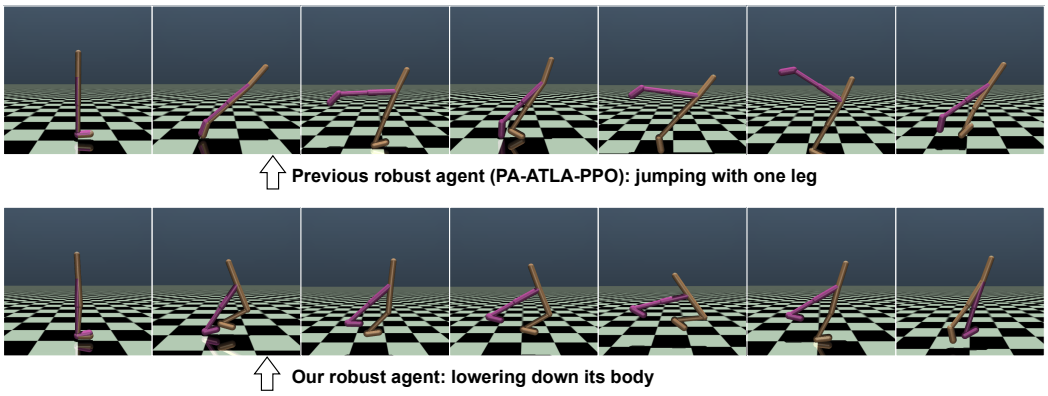


Figure 3.2: The robust Walker agents trained with **(top)** the state-of-the-art method PA-ATLA-PPO [8] and **(bottom)** our WocaR-RL. Although PA-ATLA-PPO agent also achieves high reward under attacks, it learns to jump with one leg, which is counter-intuitive and may indicate some level of overfitting to a specific attacker. In contrast, our WocaR-RL agent learns to lower down its body, which is more intuitive and interpretable. The full agent trajectories in Walker and other environments are provided in supplementary materials as GIF figures.

3.2 Related Work

Defending against Adversarial Perturbations on State Observations. **(1)** *Regularization-based methods* [19, 39, 50] enforce the policy to have similar outputs under similar inputs, which achieves certifiable performance for DQN in some Atari games. But in continuous control tasks, these methods may not reliably improve the worst-case performance. A recent work by Korkmaz [51] points out that these adversarially trained models may still be sensible to new perturbations. **(2)** *Attack-driven methods* train DRL agents with adversarial examples. Some early works [24, 52, 53, 54] apply weak or strong gradient-based attacks on state observations to train

RL agents against adversarial perturbations. Zhang et al. [18] propose Alternating Training with Learned Adversaries (ATLA), which alternately trains an RL agent and an RL adversary and significantly improves the policy robustness in continuous control games. Sun et al. [8] further extend this framework to PA-ATLA with their proposed more advanced RL attacker PA-AD. Although ATLA and PA-ATLA achieve strong empirical robustness, they require training an extra RL adversary that can be computationally and sample expensive. **(3)** There is another line of work studying *certifiable robustness* of RL policies. Several works [37, 38, 39] computed lower bounds of the action value network Q^π to certify robustness of action selection at every step. However, these bounds do not consider the distribution shifts caused by attacks, so some actions that appear safe for now can lead to extremely vulnerable future states and low long-term reward under future attacks. Moreover, these methods cannot apply to continuous action spaces. Kumar et al. and Wu et al.[55, 56] both extend randomized smoothing [57] to derive robustness certificates for trained policies. But these works mostly focus on theoretical analysis, and effective robust training approaches rather than robust training.

Adversarial Defenses against Other Adversarial Attacks. Besides observation perturbations, attacks can happen in many other scenarios. For example, the agent's executed actions can be perturbed [27, 28, 29, 30]. Moreover, in a multi-agent game, an agent's behavior can create adversarial perturbations to a victim agent [25]. Pinto et al. [36] model the competition between the agent and the attacker as a zero-sum two-player game, and train the agent under a learned attacker to tolerate both environment shifts and adversarial disturbances. We point out that although we mainly consider state adversaries, our WocaR-RL can be extended to action attacks naturally, as formulated in Section 3.7.3.1. Note that we focus on robustness against test-time attacks, different from poisoning attacks which alter the RL training process [31, 32, 33, 34, 35].

Safe RL and Risk-sensitive RL. There are several lines of work that study RL under safety/risk constraints [58, 59, 60, 61, 62] or under intrinsic uncertainty of environment dynamics [63, 64]. However, these works do not deal with adversarial attacks, which can be adaptive to the learned policy. More comparison between these methods and our proposed method is discussed in Section 3.4.

3.3 Background and Problem Setup

Test-time Adversarial Attacks. After training, the agent is deployed into the environment and executes a pre-trained fixed policy π. An attacker/adversary, during the deployment of the agent, may perturb the state observation of the agent/victim at every time step with a certain attack budget ϵ. Note that the attacker only perturbs the inputs to the policy, and the underlying state in the environment does not change. This is a realistic setting because real-world observations can come from noisy sensors or be manipulated by malicious attacks. For example, an auto-driving car receives sensory observations; an attacker may add imperceptible noise to the camera, or perturb the GPS signal, although the underlying environment (the road) remains unchanged. In this chapter, we consider the ℓ_p *thread model* which is widely used in adversarial learning literature: at step t, the attacker alters the observation s_t into $\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)$, where $\mathcal{B}_\epsilon(s_t)$ is a ℓ_p norm ball centered at s_t with radius ϵ. The above setting (ℓ_p constrained observation attack) is the same with many prior works [4, 8, 18, 19, 54].

Worst-case Return under Attacks. The natural value V^π measures how much future reward the agent gets if it follows π at all following steps. However, when there are adversarial attacks, the total reward may no longer be V^π. For example, at step t, if the attacker perturbs

the clean state s_t to $\tilde{s}_t$, then the agent will take action $\tilde{a}_t \sim \pi(\tilde{s}_t)$ instead of $a_t \sim \pi(s_t)$ and thus will get reward $R(s_t, \tilde{a}_t)$ instead of $R(s_t, a_t)$. Among all ℓ_p attackers with attack radius ϵ, it is important to identify the *strongest attacker* which minimizes the total reward of the victim agent. As justified by [19] and [8], for any given victim policy π, there exists an optimal deterministic attacker function, denoted as $h^* : \mathcal{S} \to \mathcal{B}_\epsilon(\mathcal{S})$, where $\mathcal{B}_\epsilon(\mathcal{S})$ is the ϵ-neighborhood of state space $\mathcal{S}$. Such an attacker can be learned with RL algorithms [8, 18]. Therefore, the agent's performance can be lower bounded by its total reward under h^*. We define the *worst-case value* as $\underline{V}^\pi := \mathbb{E}_{P,\tilde{a}_t \sim \pi(h^*(s_t))} \left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \tilde{a}_t\right) \mid s_0 = s\right]$. It can be shown [8] that a well-trained agent may get very low return under strong attacks. That is, $\underline{V}^\pi \ll V^\pi$.

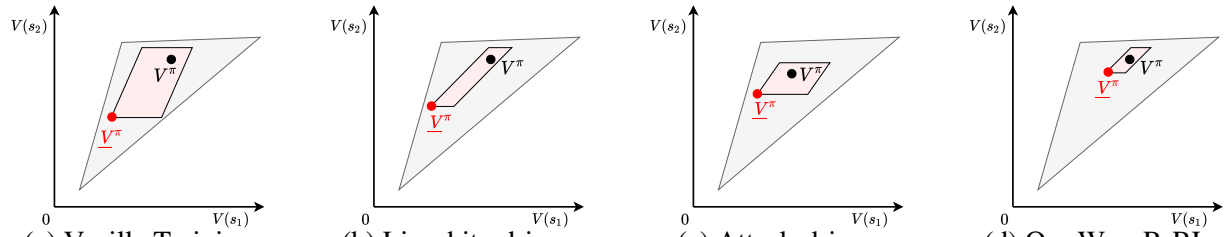
3.4 Proposed Approach

3.4.1 A Closer Look at Robust RL

In real-world applications where observations may be noisy or perturbed, it is important to ensure that the agent not only makes good decisions, but also makes safe decisions.

Existing Robust RL Approaches. There are many existing robust training methods for RL, and we summarize the common ideas as the following two categories.

(1) *Lipschitz-driven methods:* encourage the policy to output similar actions for any pair of clean state and perturbed state, i.e., $\min_\theta \max_{s \in \mathcal{S}, \tilde{s} \in \mathcal{B}_\epsilon(s)} \mathsf{Dist}(\pi_\theta(s), \pi_\theta(\tilde{s}))$, where Dist can be any distance metric. Therefore, the policy function (network) has small local Lipschitz constant at each clean state. Note that this idea is similar to many certifiable robust training methods [66] in supervised learning. For example, Fischer et al. [37] achieve provable robustness for DQN by applying the DiffAI [67] approach, so that the DQN agent selects the same action for any el-



(a) Vanilla Training (b) Lipschitz-driven (c) Attack-driven (d) Our WocaR-RL

Figure 3.3: Geometric understanding of different training methods following the polytope theory by [65] and [8]. x, y axes represent the policy value for $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}$. The grey polytope depicts the value space of all policies, while the pink polytope (referred to as value perturbation polytope) contains the values of policy π under all attacks with given constraint (ϵ-radius ℓ_p perturbations on the state input to the policy).
V^π denotes the value of a learned policy, and $\underline{V}^\pi$ stands for the worst-attack value of this policy π (located at the bottom leftmost vertex of the value perturbation polytope).
Two relations between the value perturbation polytope and policy robustness: The more distant the pink value perturbation polytope's bottom leftmost vertex is from the origin, the higher worst-attack value π has. The smaller the pink value perturbation polytope is, the less vulnerable the policy is (i.e., an ϵ-bounded state perturbation can not lead to a drastic change of the policy value).
Our method: WocaR-RL makes a policy more robust via worst-attack value estimation, worst-case-aware policy optimization and value-enhanced state regularization, which shrink the value perturbation polytope and move the value perturbation polytope's bottom leftmost vertex away from the origin.

ement inside $\mathcal{B}_\epsilon(s)$. Zhang et al. [19] propose to minimize the total variance between $\pi(s)$ and $\pi(\tilde{s})$ using convex relaxations of NNs. Although Lipschitz-driven methods are relatively efficient in training, they usually treat all states equally, and do not explicitly consider long-term rewards. Therefore, it is hard to obtain a non-vacuous reward certification, especially in continuous-action environments.

(2) *Attack-driven methods:* train the agent under adversarial attacks, which is analogous to Adversarial Training (AT) [68]. However, different from AT, a PGD attacker may not induce a robust policy in an RL problem due to the uncertainty and complexity of the environment. Zhang et al. [18] propose to alternately train an agent and an RL-based "optimal" adversary, so that the agent can adapt to the worst-case input perturbation. Therefore, attack-driven method can be formulated as $\max_\theta \underline{V}^{\pi_\theta}$. Zhang et al. [18] and a follow-up work by Sun et al. [8] apply the alternate training approach and obtain state-of-the-art robust performance. However, learning the optimal attacker using RL algorithms doubles the learning complexity and the required samples, making it hard to apply these methods to large-scale problems. Moreover, although these attack-driven methods improve the worst-case performance of an agent, the natural reward can be sacrificed.

Note that we discuss methods that improve the robustness of deep policies during training. Therefore, the focus is different from some important works [38, 55, 56] that directly use non-robust policies and execute them in a robust way.

Our Motivation: Geometric Understanding of Robust RL. The robustness of a learned RL policy can be understood from a geometric perspective. Dadashi et al. [65] point out that the value functions of all policies in a finite MDP form a polytope, as shown by the grey area in Figure 3.3. Sun et al. [8] further find that $V^{\tilde{\pi}}$, possible values of a policy π under all ϵ-constrained ℓ_p perturbations, also form a polytope (pink area in Figure 3.3), which we refer to as the *value*

perturbation polytope. Recall that in robust RL, we pursue a high natural value V^{π}, and a high worst-case value $\underline{V}^{\pi}$ which is the lower leftmost vertex of the value perturbation polytope. A vulnerable policy that outputs a different action for a perturbed state as a larger value perturbation polytope. Lipschitz-driven methods, as Figure 3.3(a) shows, attempts to shrink the size of the value perturbation polytope, but does not necessarily result in a high $\underline{V}^{\pi}$. Attack-driven methods, as Figure 3.3 shows, improves $\underline{V}^{\pi}$, but have no control over the size of the value perturbation polytope, and may not obtain a high natural value V^{π}.

Our Proposed Robust RL Principle. In contrast to prior Lipschitz-driven methods and Attack-driven methods, we propose to both "lift the position" and "shrink the size" of the value perturbation polytope. To achieve the above principle in an efficient way, we propose to (1) directly estimate and optimize the worst-case value of a policy without training the optimal attacker (worst-attack value estimation and worst-case-aware policy optimization mechanisms of WocaR-RL), and (2) regularize the local Lipschitz constants of the policy with value-enhanced weights (value-enhanced state regularization mechanism of WocaR-RL). See Section 3.4.2 for more details of the proposed algorithm.

3.4.2 WocaR-RL: Worst-case-aware Robust RL

In this section, we present *Worst-case-aware Robust RL (WocaR-RL)*, a generic framework that can be fused with any DRL approach to improve the adversarial robustness of an agent. We will introduce the three key mechanisms in WocaR-RL: worst-attack value estimation, worst-case-aware policy optimization, and value-enhanced state regularization, respectively. Then, we will illustrate how to incorporate these mechanisms into existing DRL algorithms to improve

their robustness.

Mechanism 1: Worst-attack Value Estimation

Traditional RL aims to learn a policy with the maximal value V^π. However, in a real-world problem where observations can be noisy or even adversarially perturbed, it is not enough to only consider the natural value V^π and Q^π. As motivated in Figure 3.1, two policies with similar natural rewards can get totally different rewards under attacks. To comprehensively evaluate how good a policy is in an adversarial scenario and to improve its robustness, we should be aware of the lowest possible long-term reward of the policy when its observation is adversarially perturbed with a certain attack budget ϵ at every step (with an ℓ_p attack model introduced in Section 3.3).

The worst-case value of a policy is, by definition, the cumulative reward obtained under the optimal attacker. As justified by prior works [8, 19], for any given victim policy π and attack budget $\epsilon > 0$, there exists an optimal attacker, and finding the optimal attacker is equivalent to learning the optimal policy in another MDP. We denote the optimal (deterministic) attacker's policy as h^*. However, learning such an optimal attacker by RL algorithms requires extra interaction samples from the environment, due to the unknown dynamics. Moreover, learning the attacker by RL can be hard and expensive, especially when the state observation space is high-dimensional.

Instead of explicitly learning the optimal attacker with a large amount of samples, we propose to directly estimate the worst-case cumulative reward of the policy by characterizing the vulnerability of the given policy. We first define the *worst-attack action value* of policy π as $\underline{Q}^\pi(s,a) := \mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(h^*(s_t))\right) \mid s_0 = s, a_0 = a]$. The *worst-attack value* $\underline{V}^\pi$ can be defined using h^* in the same way, as shown in Definition 32 in Section 3.7.1. Then, we introduce a novel operator $\underline{\mathcal{T}}^\pi$, namely the *worst-attack Bellman operator*, defined as below.

Definition 29 (Worst-attack Bellman Operator)**.** *For MDP $\mathcal{M}$, given a fixed policy π and attack radius ϵ, define the worst-attack Bellman operator $\underline{\mathcal{T}}^{\pi}$ as*

$$\left(\underline{\mathcal{T}}^{\pi} Q\right)(s,a) := \mathbb{E}_{s' \sim P(s,a)}[R(s,a) + \gamma \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q\left(s', a'\right)], \tag{3.1}$$

where $\forall s \in \mathcal{S}$, $\mathcal{A}_{\text{adv}}(s,\pi)$ is defined as

$$\mathcal{A}_{\text{adv}}(s,\pi) := \{a \in \mathcal{A} : \exists \tilde{s} \in \mathcal{B}_{\epsilon}(s) \text{ s.t. } \pi(\tilde{s}) = a\}. \tag{3.2}$$

Here $\mathcal{A}_{\text{adv}}(s',\pi)$ denotes the set of actions an adversary can mislead the victim π into selecting by perturbing the state s' into a neighboring state $\tilde{s} \in \mathcal{B}_{\epsilon}(s')$. This hypothetical perturbation to the *future* state s' is the key for characterizing the worst-case long-term reward under attack. The following theorem associates the worst-attack Bellman operator and the worst-attack action value.

Theorem 30 (Worst-attack Bellman Operator and Worst-attack Action Value)**.** *For any given policy π, $\underline{\mathcal{T}}^{\pi}$ is a contraction whose fixed point is $\underline{Q}^{\pi}$, the worst-attack action value of π under any ℓ_p observation attacks with radius ϵ.*

Theorem 30 proved in Section 3.7.1 suggests that the lowest possible cumulative reward of a policy under bounded observation attacks can be computed by worst-attack Bellman operator. The corresponding worst-attack value $\underline{V}^{\pi}$ can be obtained by $\underline{V}^{\pi}(s) = \min_{a \in \mathcal{A}_{\text{adv}}(s,\pi)} \underline{Q}^{\pi}(s,a)$.

How to Compute $\mathcal{A}_{\text{adv}}$. To obtain $\mathcal{A}_{\text{adv}}(s,\pi)$, we need to identify the actions that can be the outputs of the policy π when the input state s is perturbed within $\mathcal{B}_{\epsilon}(s)$. This can be solved by commonly-used convex relaxation of neural networks [66, 69, 70, 71, 72], where layer-wise lower and upper bounds of the neural network are derived. That is, we calculate $\overline{\pi}$ and $\underline{\pi}$ such that $\overline{\pi}(s) \geq \pi(\hat{s}) \geq \underline{\pi}(s), \forall \hat{s} \in \mathcal{B}_{\epsilon}(s)$. With such a relaxation, we can obtain a superset of $\mathcal{A}_{\text{adv}}$, namely $\hat{\mathcal{A}}_{\text{adv}}$. Then, the fixed point of Equation (3.1) with $\mathcal{A}_{\text{adv}}$ being replaced by $\hat{\mathcal{A}}_{\text{adv}}$ becomes

a lower bound of the worst-attack action value. For a continuous action space, $\hat{\mathcal{A}}_{\text{adv}}(s,\pi)$ contains actions bounded by $\overline{\pi}(s)$ and $\underline{\pi}(s)$. For a discrete action space, we can first compute the maximal and minimal probabilities of taking each action, and derive the set of actions that are likely to be selected. The computation of $\hat{\mathcal{A}}_{\text{adv}}$ is not expensive, as there are many efficient convex relaxation methods [67, 72] which compute $\overline{\pi}$ and $\underline{\pi}$ with only constant-factor more computations than directly computing $\pi(s)$. Experiment in Section 3.5 verifies the efficiency of our approach, where we use a well-developed toolbox auto_LiRPA [73] to calculate the convex relaxation.

Estimating Worst-attack Value. Note that the worst-attack Bellman operator $\underline{\mathcal{T}}^{\pi}$ is similar to the optimal Bellman operator $\mathcal{T}^*$, although it uses $\min_{a\in\mathcal{A}_{\text{adv}}}$ instead of $\max_{a\in\mathcal{A}}$. Therefore, once we identify $\mathcal{A}_{\text{adv}}$ as introduced above, it is straightforward to compute the worst-attack action value using Bellman backups. To model the worst-attack action value, we train a network named *worst-attack critic*, denoted by $\underline{Q}^{\pi}_{\phi}$, where ϕ is the parameterization. Concretely, for any mini-batch $\{s_t,a_t,r_t,s_{t+1}\}_{t=1}^{N}$, $\underline{Q}^{\pi}_{\phi}$ is optimized by minimizing the following estimation loss:

$$\mathcal{L}_{\text{est}}(\underline{Q}^{\pi}_{\phi}) := \frac{1}{N}\sum_{t=1}^{N}(\underline{y}_t - \underline{Q}^{\pi}_{\phi}(s_t,a_t))^2, \text{where } \underline{y}_t = r_t + \gamma \min_{\hat{a}\in\mathcal{A}_{\text{adv}}(s_{t+1},\pi)} \underline{Q}^{\pi}_{\phi}(s_{t+1},\hat{a}). \tag{3.3}$$

For a discrete action space, $\mathcal{A}_{\text{adv}}$ is a discrete set and solving $\underline{y}_t$ is straightforward. For a continuous action space, we use gradient descent to approximately find the minimizer $\hat{a}$. Since $\mathcal{A}_{\text{adv}}$ is in general small, this minimization is usually easy to solve. In MuJoCo, we find that 50-step gradient descent already converges to a good solution with little computational cost, as detailed in Section 3.7.4.5.

Differences with Worst-case Value Estimation in Related Work. Our proposed worst-attack Bellman operator is different from the worst-case Bellman operator in the literature of risk-sensitive RL [58, 59, 60, 61, 62, 74], whose goal is to avoid unsafe trajectories under the intrinsic

uncertainties of the MDP. These inherent uncertainties of the environment are independent of the learned policy. In contrast, our focus is to defend against adversarial perturbations created by malicious attackers that can be *adaptive* to the policy. The GWC reward proposed by [39] also estimates the worst-case reward of a policy under state perturbations. But their evaluation is based on a greedy strategy and requires interactions with the environment, which is different from our estimation.

Mechanism 2: Worst-case-aware Policy Optimization

So far we have introduced how to evaluate the worst-attack value of a policy by learning a worst-attack critic. Inspired by the actor-critic framework, where the actor policy network π_θ is optimized towards a direction that the critic value increases the most, we can regard worst-attack critic as a special critic that directs the actor to increase the worst-attack value. That is, we encourage the agent to select an action with a higher worst-attack action value, by minimizing the worst-attack policy loss below:

$$\mathcal{L}_{\text{wst}}(\pi_\theta; \underline{Q}^{\pi}_{\phi}) := -\frac{1}{N}\sum_{t=1}^{N}\sum_{a\in\mathcal{A}}\pi_\theta(a|s_t)\underline{Q}^{\pi}_{\phi}(s_t, a), \tag{3.4}$$

where $\underline{Q}^{\pi}_{\phi}$ is the worst-attack critic learned via $\mathcal{L}_{\text{est}}$ introduced in Equation (3.3). Note that $\mathcal{L}_{\text{wst}}$ is a general form, while the detailed implementation of the worst-attack policy optimization can vary depending on the architecture of π_θ in the base RL algorithm (e.g. PPO has a policy network, while DQN acts using the greedy policy induced by a Q network). In Section 3.7.2.2 and Section 3.7.2.3, we illustrate how to implement $\mathcal{L}_{\text{wst}}$ for PPO and DQN as two examples.

The proposed worst-case-aware policy optimization has several **merits** compared to prior ATLA [18] and PA-ATLA [8] methods which alternately train the agent and an RL attacker. **(1)** Learning the optimal attacker h^* requires collecting extra samples using the current policy

(on-policy estimation). In contrast, $\underline{Q}^{\pi}_{\phi}$ can be learned using off-policy samples, e.g., historical samples in the replay buffer, and thus is more suitable for training where the policy changes over time. ($\underline{Q}^{\pi}_{\phi}$ depends on the current policy via the computation of $\mathcal{A}_{\text{adv}}$.) **(2)** We properly exploit the policy function that is being trained by computing the set of possibly selected actions $\hat{\mathcal{A}}_{\text{adv}}$ for any state. In contrast, ATLA [18] learns an attacker by treating the current policy as a black box, ignoring the intrinsic properties of the policy. PA-ATLA [8], although assumes white-box access to the victim policy, also needs to explore and learn from extra on-policy interactions. **(3)** The attacker trained with DRL methods, namely $\hat{h}^*$, is not guaranteed to converge to an optimal solution, such that the performance of π estimated under $\hat{h}^*$ can be overly optimistic. Our estimation, as mentioned in Mechanism 1, computes a lower bound of $\underline{Q}^{\pi}$ and thus can better indicate the robustness of a policy.

Mechanism 3: Value-enhanced State Regularization

As discussed in Section 3.1, the vulnerability of a deep policy comes from both the policy's intrinsic vulnerability with the RL dynamics and the DNN approximator. The first two mechanisms of WocaR-RL mainly focus on the policy's intrinsic vulnerability, i.e., let the policy select actions that are less vulnerable to possible attacks in all future steps. However, if a bounded state perturbation can cause the network to output a very different action, then the $\mathcal{A}_{\text{adv}}$ set will be large and $\underline{Q}^{\pi}$ can thus be low. Therefore, it is also important to encourage the trained policy to output similar actions for the clean state s and any $\tilde{s} \in \mathcal{B}_{\epsilon}(s)$, as is done in prior work [19, 37, 50]. But different from these prior methods, we note that different states should be treated differently. Some states are "critical" where selecting a bad action will result in catastrophic consequences. For example, when the agent gets close to the bomb in Figure 3.1, we should make the network more resistant to adversarial state perturbations. To differentiate states based on their impacts on

future reward, we propose to measure the importance of states with Definition 31 below.

Definition 31 (State Importance Weight)**.** *Define state importance weight of $s \in \mathcal{S}$ for policy π as*

$$w(s) = \max_{a_1 \in \mathcal{A}} Q^\pi(s, a_1) - \min_{a_2 \in \mathcal{A}} Q^\pi(s, a_2). \tag{3.5}$$

To justify whether Definition 31 can characterize state importance, we train a DQN network in an Atari game Pong, and show the states with the highest weight and the lowest weight in Figure 3.4, among many state samples. We can see that the state with higher weight in Figure 3.4(left) is indeed crucial for the game, as the green agent paddle is close to the ball. Conversely, a less-important state in Figure 3.4(right) does not have significantly different future rewards under different actions. Computing $w(s)$ is easy in a discrete action space, while in a continuous action space, one can use gradient descent to approximately find the maximal and the minimal Q values for a state. Similar to the computation of Equation (3.3) with a continuous action space, we find that a 50-step gradient descent works well in experiments.

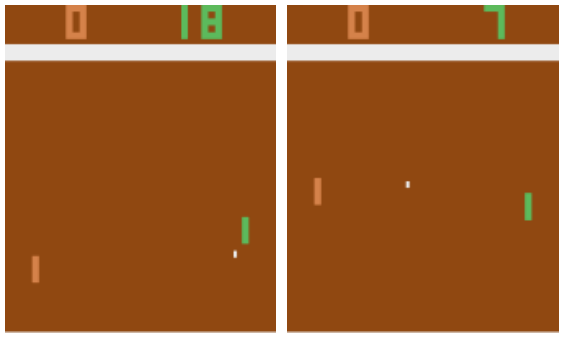


Figure 3.4: States in Pong with **(left)** high weight $w(s)$ and **(right)** low weight $w(s)$.

By incorporating the state importance weight $w(s)$, we regularize the policy network and let it pay more attention to more crucial states, by minimizing the following loss:

$$\mathcal{L}_{\text{reg}}(\pi_\theta) = \frac{1}{N} \sum_{t=1}^{N} w(s_t) \max_{\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)} \mathsf{Dist}(\pi_\theta(s_t), \pi_\theta(\tilde{s}_t)), \tag{3.6}$$

where Dist can be any distance measure between two distributions (e.g., KL-divergence). Minimizing $\mathcal{L}_{\text{reg}}$ can result in a smaller $\mathcal{A}_{\text{adv}}$, and thus the worst-attack value will be closer to the

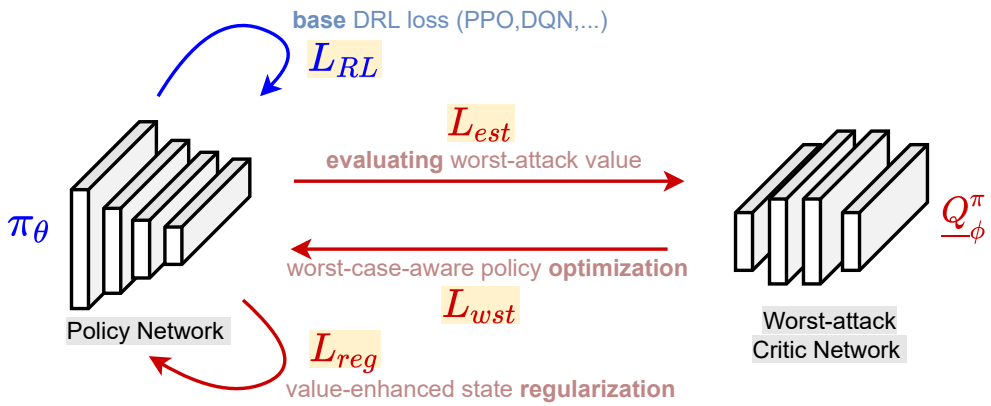


Figure 3.5: Training architecture of WocaR-RL. (Components proposed in this chapter are colored as red.)

natural value.

WocaR-RL: A Generic Robust Training Framework

So far we have introduced three key mechanisms and their loss functions, $\mathcal{L}_{\text{est}}$ in Equation (3.3), $\mathcal{L}_{\text{wst}}$ in Equation (3.4) and $\mathcal{L}_{\text{reg}}$ in Equation (3.6). Then, our robust training framework WocaR-RL combines these losses with any base RL algorithm. To be more specific, as shown in Figure 3.5, for any base RL algorithm that trains policy π_θ using loss $\mathcal{L}_{\text{RL}}$, we learn an extra worst-attack critic network $\underline{Q}^{\pi}_{\phi}$ by minimizing

$$\mathcal{L}_{\underline{Q}^{\pi}_{\phi}} := \mathcal{L}_{\text{est}}(\underline{Q}^{\pi}_{\phi}), \tag{3.7}$$

and combine $\mathcal{L}_{\text{wst}}$ and $\mathcal{L}_{\text{reg}}$ with $\mathcal{L}_{\text{RL}}$ to optimize π_θ by minimizing

$$\mathcal{L}_{\pi_\theta} := \mathcal{L}_{\text{RL}}(\pi_\theta) + \kappa_{\text{wst}}\mathcal{L}_{\text{wst}}(\pi_\theta; \underline{Q}^{\pi}_{\phi}) + \kappa_{\text{reg}}\mathcal{L}_{\text{reg}}(\pi_\theta), \tag{3.8}$$

where κ_{wst} and κ_{reg} are hyperparameters balancing between natural performance and robustness. Note that $\underline{Q}^{\pi}_{\phi}$ is trained together but independently with π_θ using historical transition samples, so WocaR-RL does not require extra samples from the environment. WocaR-RL can also be interpreted from a geometric perspective based on prior RL polytope theory [8, 65] as detailed in Section 3.4.1.

Our WocaR-RL is a generic robust training framework that can be used to robustify existing DRL algorithms. We provide two case studies: **(1)** combining WocaR-RL with a policy-based algorithm PPO [16], namely *WocaR-PPO*, and **(2)** combining WocaR-RL with a value-based algorithm DQN [15], namely *WocaR-DQN*. The pseudocodes of WocaR-PPO and WocaR-DQN are illustrated in Section 3.7.2.2 and Section 3.7.2.3. The application of WocaR-RL to other DRL methods is then straightforward, since most DRL methods are either policy-based or value-based. Next, we show by experiments that WocaR-PPO and WocaR-DQN achieve state-of-the-art robustness with superior efficiency, in various continuous control tasks and video game environments. We also empirically verify the effectiveness of each of the 3 mechanisms of WocaR-RL and their weights by ablation study in Section 3.5.2.

3.5 Empirical Results

In this section, our experimental evaluations on various MuJoCo and Atari environments aim to study the following questions:

(1) Can WocaR-RL learn policies with better **robustness** under existing strong adversarial attacks?

(2) Can WocaR-RL maintain **natural performance** when improving robustness?

(3) Can WocaR-RL learn more **efficiently** during robust training?

(4) Is each mechanism in WocaR-RL **effective**?

Problem (1), (2) and (3) are answered in Section 3.5.1 with detailed empirical results, and problem (4) is studied in Section 3.5.2 via ablation experiments.

3.5.1 Experiments and Evaluations

Environments. Following most prior works [18, 19, 39] and the released implementation, we apply our WocaR-RL to PPO [16] on 4 MuJoCo tasks with continuous action spaces, including Hopper, Walker2d, Halfcheetah and Ant, and to DQN [15] agents on 4 Atari games including Pong, Freeway, BankHeist and RoadRunner, which have high dimensional pixel inputs and discrete action spaces.

Baselines and Implementation. We compare our algorithm with several state-of-the-art robust training methods, including (1) *SA-PPO/SA-DQN* [19]: regularizing policy networks by convex relaxation. (2) *ATLA-PPO* [18]: alternately training an agent and an RL attacker. (3) *PA-ATLA-PPO* [8]: alternately training an agent and a more advanced RL attacker PA-AD. (4) *RADIAL-PPO/RADIAL-DQN* [39]: optimizing policy network by designed adversarial loss functions based on robustness bounds. SA and RADIAL have both PPO and DQN versions, which are compared with our WocaR-PPO and WocaR-DQN. But ATLA and PA-ATLA do not provide DQN versions, since alternately training on DQN can be expensive as explained in the original papers [8]. (PA-ATLA has an A2C version, which we compare in Section 3.7.4.2.) Therefore, we reproduce their ATLA-PPO and PA-ATLA-PPO results and compare them with our WocaR-PPO. More implementation and hyperparameter details are provided in Section 3.7.4.1.

Case I: Robust PPO for MuJoCo Continuous Control

Evaluation Metrics. To reflect both the natural performance and robustness of trained agents, we report the average episodic rewards under no attack and against various attacks. For a comprehensive robustness evaluation, we attack the trained robust models with multiple existing attack methods, including: (1) *MaxDiff* [19] (maximal action difference), (2) *Robust Sarsa (RS)*

[19] (attacking with a robust action-value function), (3) *SA-RL* [19] (finding the optimal state adversary) and (4) *PA-AD* [8] (the *existing strongest attack* by learning the optimal policy adversary with RL). For a clear comparison, we use the same attack radius ϵ as in most baselines [8, 18, 19].

Performance and Robustness of WocaR-PPO Figure 3.6 (left four columns) shows performance curves during training under four different adversarial attacks. Among all four attack algorithms, WocaR-PPO converges much faster than baselines, and often achieves the best asymptotic robust performance, especially under the strongest PA-AD attack. It is worth emphasizing that since we train a robust agent without explicitly learning an RL attacker, our method not only obtains stronger robustness and much higher efficiency, but also a more general defense: WocaR-PPO obtains comprehensively superior performance against a variety of attacks compared against existing SOTA algorithms based on learned attackers (ATLA-PPO, PA-ATLA-PPO). Additionally, in our experiments, WocaR-PPO learns relatively more universal defensive behaviors as shown in Figure 3.2, which can physically explain why our algorithm can defend against diverse attacks. We provide policy demonstrations in multiple tasks in our supplementary materials.

The comparison of natural performance and the worst-case performance appears in Figure 3.6 (right). We see that WocaR-PPO maintains competitive natural rewards under no attack compared with other baselines, which demonstrates that our algorithm gains more robustness without losing too much natural performance. The full results of baselines and our algorithm under different attack evaluations are provided by Table 3.2 in Section 3.7.4.2 (including performance under random attacks).

Efficiency of Training WocaR-PPO. The learning curves in Figure 3.6 (left) directly show the sample efficiency of WocaR-PPO. Following the optimal settings provided in [18, 19,

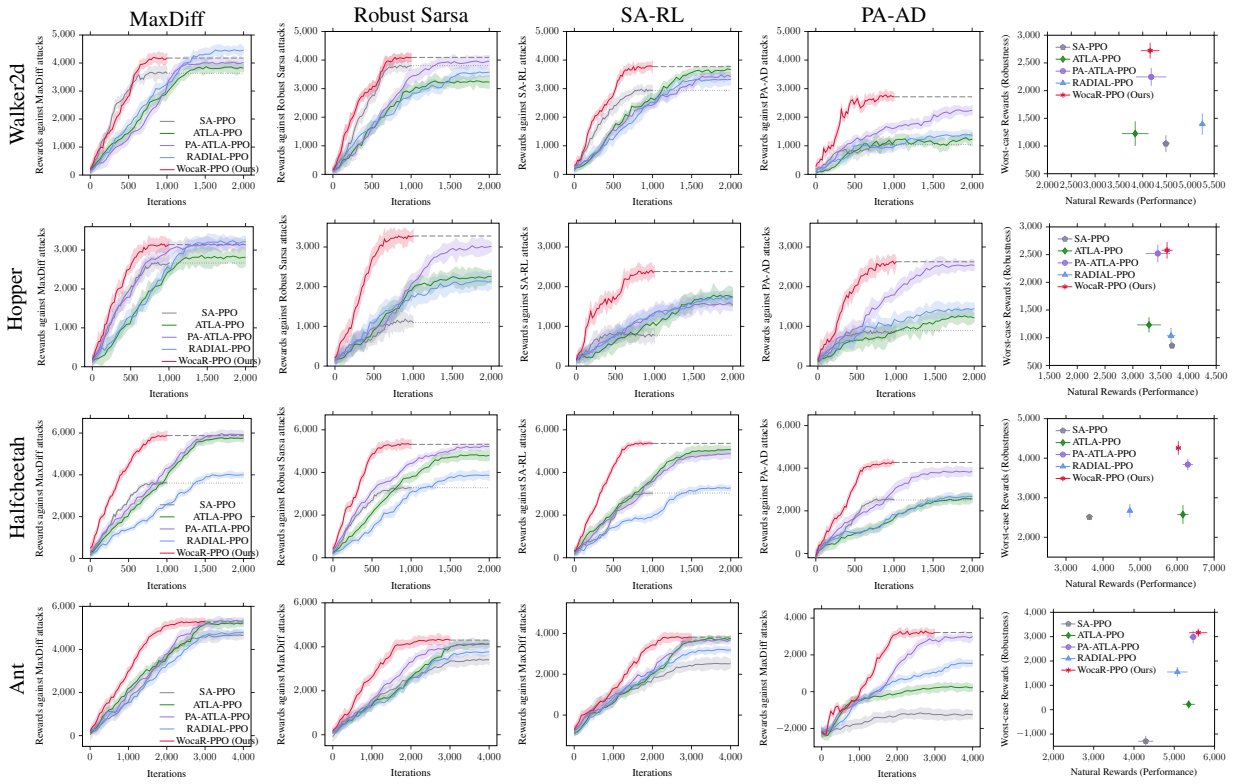


Figure 3.6: **Robustness, Efficiency and High Natural Performance of WocaR-PPO. (Left four columns)** Learning curves of rewards under MaxDiff, Robust Sarsa, SA-RL and PA-AD *(the strongest)* attacks during training on four environments. **(Rightmost column)** Average episode natural rewards v.s. average worst rewards under attacks. Each row shows the performance of baselines and WocaR-PPO on one environment. Shaded regions are computed over 20 random seeds. Results under more attack radius ϵ's are in Section 3.7.4.3.

39], our method takes *50% training steps* required by RADIAL-PPO and ATLA methods on Hopper, Walker2d, and Halfcheetah because RADIAL-PPO needs more steps to ensure convergence and ATLA methods require additional adversary training steps. When solving high dimensional environments like Ant, WocaR-PPO only requires *75% steps* compared with all other baselines to converge. We also provide additional results of baselines using the same training steps as WocaR-PPO in Section 3.7.4.4.

In terms of time efficiency, WocaR-PPO saves *50% training time* for convergence on Hopper, Walker2d, and Halfcheetah, and *32% time* on Ant compared with the SOTA method. Therefore,

WocaR-PPO achieves both higher computational efficiency and higher sample efficiency than SOTA baselines. Detailed costs in time and sampling are in Section 3.7.4.5.

Case II: Robust DQN for Atari Video Games

Evaluation Metrics. Since Atari games have pixel state spaces and discrete action spaces, the applicable attacking algorithms also differ from those in MuJoCo tasks. We include the following common attacks: (1) 10-step untargeted *PGD* (projected gradient descent) attack, (2) *MinBest* [4], which minimizes the probability of choosing the "best" action, (3) *PA-AD* [8], as the state-of-the-art RL-based adversarial attack algorithm.

Performance and Robustness of WocaR-DQN. Table 3.1 presents the results on four Atari games under attack radius $\epsilon = 3/255$, while results and analysis under smaller attack radius $1/255$ are in Section 3.7.4.2. We can see that *our WocaR-DQN consistently outperforms baselines under MinBest and PA-AD attacks in all environments, with a significant advance under the strongest (worst-case) PA-AD attacks compared with other robust agents.* Under PGD attacks, WocaR-DQN performs comparably with the state-of-the-art in Freeway and Pong (which are simpler games) and gains higher rewards than other agents in BankHeist and Roadrunner. Since SA-DQN and RADIAL-DQN focus on bounding and smoothing the policy network and do not consider the policy's intrinsic vulnerability, they are robust under the PGD attack but still vulnerable against the stronger PA-AD attack.

Efficiency of Training WocaR-DQN. The total training time for SA-DQN, RADIAL-DQN, and our WocaR-DQN are roughly 35, 17, and 18 hours, respectively. All baselines are trained for 6 million frames on the same hardware. Therefore, WocaR-DQN is 49% faster (and is more robust) than SA-DQN. Compared to the more advanced baseline RADIAL-DQN, although WocaR-DQN is 5% slower, it achieves better robustness (539% higher reward than RADIAL-

Model	Pong				BankHeist			
	Natural Reward	PGD	MinBest	PA-AD	Natural Reward	PGD	MinBest	PA-AD
		ϵ= 3/255				ϵ= 3/255		
DQN	21.0 ± 0.0	-21.0 ± 0.0	-9.7 ± 4.0	-19.0 ± 2.2	**1308 ± 24**	0 ± 0	119 ± 65	102 ± 92
SA-DQN	21.0 ± 0.0	21.0 ± 0.0	20.6 ± 3.5	18.7 ± 2.6	1245 ± 14	1176 ± 63	1024 ± 31	489 ± 106
RADIAL-DQN	21.0 ± 0.0	21.0 ± 0.0	19.5 ± 2.1	13.2 ± 1.8	1178 ± 4	1176 ± 63	928 ± 113	508 ± 85
WocaR-DQN (Ours)	**21.0 ± 0.0**	**21.0 ± 0.0**	**20.8 ± 3.3**	**19.7 ± 2.4**	1220 ± 12	**1214 ± 7**	**1045 ± 20**	**754 ± 102**

Model	Freeway				RoadRunner			
	Natural Reward	PGD	MinBest	PA-AD	Natural Reward	PGD	MinBest	PA-AD
		ϵ= 3/255				ϵ= 3/255		
DQN	**34.0 ± 0.1**	0.0 ± 0.0	5.5 ± 1.8	4.7 ± 2.9	**45527 ± 4894**	0 ± 0	2985 ± 1440	203 ± 65
SA-DQN	30.0 ± 0.0	30.0 ± 0.0	18.3 ± 3.0	9.5 ± 3.8	44638 ± 2367	20678 ± 1563	4214 ± 2587	5516 ± 4684
RADIAL-DQN	33.1 ± 0.2	**33.2 ± 0.2**	16.4 ± 2.3	10.8 ± 3.6	44675 ± 5854	38576 ± 1960	8476 ± 3964	1290 ± 4015
WocaR-DQN (Ours)	31.2 ± 0.4	31.4 ± 0.3	**19.8 ± 3.8**	**12.3 ± 3.2**	44156 ± 2279	**38720 ± 1765**	**10545 ± 2984**	**8239 ± 2766**

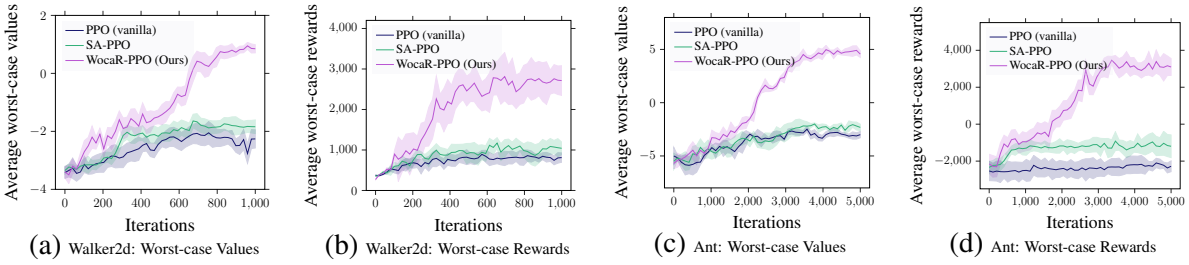
Table 3.1: **Robustness and High Natural Performance of WocaR-DQN.** Average episode rewards ± standard deviation over 50 episodes on three baselines and WocaR-DQN on four Atari environments. Best results (natural reward of under attacks for each column) on each environment boldfaced. WocaR-DQN outperforms all the baselines in most cases or gains similar performance in the other metrics. We highlight the most robust agent as gray. Each result is obtained with 10 random seeds.

DQN in RoadRunner).

3.5.2 Verifying Effectiveness of WocaR-RL

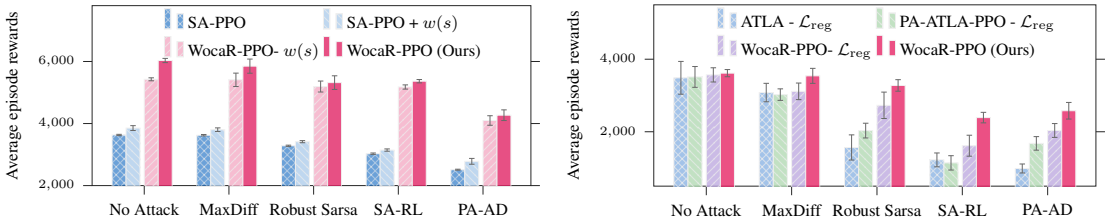
Now we dive deeper into the algorithmic design and verify the effectiveness of WocaR-RL by ablation studies on WocaR-PPO.

(1) Worst-attack value estimation. We show the learned worst-attack value estimation, $\underline{Q}^{\pi}_{\phi}$, during the training process in Figure 3.7a and 3.7c, in comparison with the actual reward under the strongest attack (PA-AD [8]) in Figure 3.7b and 3.7d. The pink curves in both plots suggest that *our worst-attack value estimation matches the trend of actual worst-case reward under attacks,* although the network estimated value and the real reward have different scales due to the commonly-used reward normalization for learning stability. Therefore, the effectiveness of our proposed worst-attack value estimation ($\mathcal{L}_{\text{est}}$) is verified.



(a) Walker2d: Worst-case Values (b) Walker2d: Worst-case Rewards (c) Ant: Worst-case Values (d) Ant: Worst-case Rewards

Figure 3.7: **(a)&(b)** Comparison between estimated worst-attack action values $\underline{Q}^{\pi}_{\phi}$ and actual worst-case rewards under the strongest attacksduring training on Walker2d; **(c)&(d)** The comparison between worst-case values and rewards to verify worst-attack value estimation on Ant.



(a) Halfcheetah: Ablation studies for $w(s)$ (b) Hopper: Ablation studies for $\mathcal{L}_{\text{reg}}$

Figure 3.8: **(a)** Ablation evaluations for state importance weight $w(s)$ under no attack and four types of attacks on Halfcheetah; **(b)** Ablation studies for state regularization $\mathcal{L}_{\text{reg}}$ under different evaluation metrics on Hopper. Ablated results on other environments are in Section 3.7.4.8.

(2) Worst-case-aware policy optimization. Compared to vanilla PPO and SA-PPO, we can see that *WocaR-PPO improves the worst-attack value and the worst-case reward during training*, suggesting the effectiveness of our worst-attack value improvement ($\mathcal{L}_{\text{wst}}$). The comparison of natural rewards, as well as curves in other environments, are provided in Section 3.7.4.6. Moreover, the adjustable weight κ_{wst} in Equation (3.8) controls the trade-off between natural value and worst-attack value in policy optimization. When κ_{wst} is high, the policy pays more attention to its worst-attack value. Section 3.7.4.7 verifies that *WocaR-RL, with different values of weight κ_{wst}, produces different robustness and natural performance while consistently dominating other robust agents*, which is another advantage of our algorithm.

(3) Value-enhanced state regularization. We conduct ablation experiments to analyze the effect of two techniques: our proposed state importance weight $w(s)$ and the state regular-

ization loss $\mathcal{L}_{\text{reg}}$ [19]. In Figure 3.8a, we compare the performance of the original WocaR-PPO to a variant of WocaR-PPO without the state importance weight $w(s)$ on Halfcheetah, which visually indicates that $w(s)$ can help agents boost the robustness. Since SA-PPO [19] also uses a state regularization technique, the improvement of SA-PPO added with $w(s)$ also show the universal effectiveness of our state importance. Without $w(s)$, our algorithm also achieves similar or better performance than baselines, but including this inexpensive technique $w(s)$ gives WocaR-RL a greater advantage, especially under learned strong attacks SA-RL and PA-AD. Figure 3.8b presents the performance of ATLA methods and our algorithm without $\mathcal{L}_{\text{reg}}$ on Hopper, which verifies that WocaR-PPO also yields the superior performance when removing the regularization technique. And the comparison between WocaR-PPO and WocaR-PPO without $\mathcal{L}_{\text{reg}}$ demonstrates that the weighted state regularization is beneficial to enhancing the robustness in our algorithm. Detailed ablation studies for $w(s)$ and $\mathcal{L}_{\text{reg}}$ on four MuJoCo environments are shown in Section 3.7.4.8.

3.6 Conclusion

This chapter proposes a robust RL training framework, WocaR-RL, that evaluates and improves the long-term robustness of a policy via worst-attack value estimation, worst-case-aware policy optimization, and value-enhanced state regularization. Different from recent state-of-the-art adversarial training methods [8, 18] which train an extra adversary to improve the robustness of an agent, we directly estimate and improve the lower bound of the agent's cumulative reward. As a result, WocaR-RL not only achieves better robustness than state-of-the-art robust RL approaches, but also halves the total sample complexity and computation complexity, in a wide

range of Atari and MuJoCo tasks.

There are several aspects to improve or extend the current approach. First, the proposed worst-attack Bellman operator in theory gives the exact worst-case value of a policy under ℓ_p bounded attacks. But in practice, it is hard to compute the set $\mathcal{A}_{\text{adv}}$ directly, so we use convex relaxation to obtain a superset of it, $\hat{\mathcal{A}}_{\text{adv}}$. As a result, the fixed point of worst-attack Bellman operator with $\mathcal{A}_{\text{adv}}$ being replaced by $\hat{\mathcal{A}}_{\text{adv}}$ is a lower bound of the worst-case value. Then, our algorithm increases the worst-case value by improving its lower bound, as visualized and explained in Figure 3.3 in Section 3.4.1. Therefore, one potential way of further improving the robustness is using a tighter relaxation. In addition, we mainly consider the ℓ_p threat model as is common in most related works. But in real-world applications, other attack models could exist (e.g. patch attacks [75]), and improving the robustness of RL agents in these scenarios is another important research direction.

3.7 Supplemental Materials: Proofs and Additional Details

3.7.1 Additional Definitions and Proofs

Similar to the worst-attack action value, we can define the worst-attack value as below:

Definition 32 (Worst-attack Value)**.** *For a given policy π, define the worst-attack value of π as*

$$\underline{V}^{\pi}(s) := \mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(h^*(s_t))\right) \mid s_0 = s], \tag{3.9}$$

where h^* is the optimal attacker which minimizes the victim's cumulative reward under the ϵ constraint.

Proof of Theorem 30. First, we show that $\underline{\mathcal{T}}^{\pi}$ is a contraction.

For any two Q functions $Q_1 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we have

$$
\begin{aligned}
&\left\|\underline{\mathcal{T}}^{\pi} Q_1 - \underline{\mathcal{T}}^{\pi} Q_2\right\|_{\infty} \\
&= \max_{s,a} \left| \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left[R(s,a) + \gamma \min_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} Q_1\left(s', a'\right) - R(s,a) + \gamma \min_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} Q_2\left(s', a'\right) \right] \right| \\
&= \gamma \max_{s,a} \left| \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left[\min_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} Q_1\left(s', a'\right) - \min_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} Q_2\left(s', a'\right) \right] \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left| \min_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} Q_1\left(s', a'\right) - \min_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} Q_2\left(s', a'\right) \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \max_{a' \in \mathcal{A}_{\mathrm{adv}}(s',\pi)} \left| Q_1\left(s', a'\right) - Q_2\left(s', a'\right) \right| \\
&= \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left\| Q_1 - Q_2 \right\|_{\infty} \\
&= \gamma \left\| Q_1 - Q_2 \right\|_{\infty}
\end{aligned}
$$

The second inequality comes from the fact that,

$$
\left| \min_{x_1} f(x_1) - \min_{x_2} g(x_2) \right| \leq \max_{x} \left| f(x) - g(x) \right|
$$

The operator $\underline{\mathcal{T}}^{\pi}$ satisfies,

$$
\left\|\underline{\mathcal{T}}^{\pi} Q_1 - \underline{\mathcal{T}}^{\pi} Q_2\right\|_{\infty} \leq \gamma \left\| Q_1 - Q_2 \right\|_{\infty}
$$

so it is a contraction in the sup-norm.

Recall the definition of worst-attack action value:

$$
\underline{Q}^{\pi}(s,a) := \mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(h^*(s_t))\right) \mid s_0 = s, a_0 = a], \tag{3.10}
$$

where h^* is the optimal attacker which minimizes the victim's cumulative reward under the ϵ constraint. That is, the optimal attacker h^* lets the agent select the worst possible action among all achievable actions in $\mathcal{A}_{\mathrm{adv}}$. Hence, we have $\underline{Q}^{\pi}(s,a) = \underline{\mathcal{T}}^{\pi} \underline{Q}^{\pi}(s,a)$. Therefore, $\underline{Q}^{\pi}(s,a)$ is

the fixed point of the Bellman operator $\underline{\mathcal{T}}^{\pi}$.

□

3.7.2 Algorithm Details

3.7.2.1 Computing $\mathcal{A}_{\text{adv}}$ by Network Bounding Techniques

Recall that $\mathcal{A}_{\text{adv}}(s, \pi) = \{a \in \mathcal{A} : \exists \tilde{s} \in \mathcal{B}_{\epsilon}(s) \text{ s.t. } \pi(\tilde{s}) = a\}$ is the set of actions that π may be misled to select in state s. Computing the exact $\mathcal{A}_{\text{adv}}$ is difficult due to the complexity of neural networks, so we use relaxations of network such as Interval Bound Propagation (IBP) [69, 71] to approximately calculate $\mathcal{A}_{\text{adv}}$.

A Brief Introduction to Convex Relaxation Methods. Convex relaxation methods are techniques to bound a neural network that provide the upper and lower bound of the neural network output given a bounded l_p perturbation to the input. In particular, we take l_∞ as an example, which has been studied extensively in prior works. Formally, let f_θ be a real-valued function parameterized by a neural network θ, and let $f_\theta(s)$ denote the output of the neural network with the input s. Given an l_∞ perturbation budget ϵ, convex relaxation method outputs $(\underline{f_\theta(s)}, \overline{f_\theta(s)})$ such that

$$\underline{f_\theta(s)} \leq \min_{\|s'-s\|_\infty \leq \epsilon} f_\theta(s') \leq \max_{\|s'-s\|_\infty \leq \epsilon} f_\theta(s') \leq \overline{f_\theta(s)}$$

Recall that we use π_θ to denote the parameterized policy being trained that maps a state observation to a distribution over the action space, and π denotes the deterministic policy refined from π_θ with $\pi(s) = \text{argmax}_{a \in \mathcal{A}} \pi_\theta(a|s)$. $\mathcal{A}_{\text{adv}}(s, \pi)$ contains actions that could be selected by π (with the highest probability in π_θ's output) when s is perturbed within a ϵ-radius ball. Our goal is to approximately identify a superset of $\mathcal{A}_{\text{adv}}(s, \pi)$, i.e., $\hat{\mathcal{A}}_{\text{adv}}(s, \pi)$, via the convex relaxation

of networks introduced above.

Computing $\mathcal{A}_{\text{adv}}$ in Continuous Action Space. The most common policy parameterization in a continuous action space is through a Gaussian distribution. Let $\mu_\theta(s)$ be the mean of Gaussian computed by $\pi_\theta(s)$, then $\pi = \mu(s)$. Therefore, we can use network relaxation to compute an upper bound and a lower bound of μ_θ with input $\mathcal{B}_\epsilon(s)$. Then, $\hat{\mathcal{A}}_{\text{adv}}(s, \pi) = [\underline{\mu_\theta(s)}, \overline{\mu_\theta(s)}]$, i.e., a set of actions that are coordinate-wise bounded by $\underline{\mu_\theta(s)}$ and $\overline{\mu_\theta(s)}$. For other continuous distributions, e.g., Beta distribution, the computation is similar, as we only need to find the largest and smallest actions. In summary, we can compute $\hat{\mathcal{A}}_{\text{adv}}(s, \pi) = [\underline{\pi_\theta(s)}, \overline{\pi_\theta(s)}]$.

Computing $\mathcal{A}_{\text{adv}}$ in Discrete Action Space. For a discrete action space, the output of π_θ is a categorical distribution, and π selects the action with the highest probability. Or equivalently, in value-based algorithms like DQN, the Q network (can be regarded as π_θ) outputs the Q estimates for each action, and π selects the action with the highest Q value. In this case, we can compute the upper and lower bound of π_θ in every dimension (corresponding to an action), denoted as $\overline{a}_i, \underline{a}_i, \forall 1 \leq i \leq |\mathcal{A}|$. Then, an action $a_i \in \mathcal{A}$ is in $\hat{\mathcal{A}}_{\text{adv}}$ if for all $1 \leq j \leq |\mathcal{A}|, j \neq i$, we have $\overline{a}_i > \underline{a}_j$.

Implementation details of $\mathcal{A}_{\text{adv}}$ For a continuous action space, interval bound propagation (IBP) is the cheapest method to implement convex relaxation. We use IBP+Backward relaxation provided by *auto_LiRPA* library, following [19] to efficiently produce tighter bounds $\mathcal{A}_{\text{adv}}$ for the policy networks π_{theta}. For a discrete action space, we compute the layer-wise output bounds for the Q-network by applying robustness verification algorithms from [39].

3.7.2.2 Worst-case-aware Robust PPO (WocaR-PPO)

In policy-based DRL methods [16, 76, 77] such as PPO, the actor policy π_θ is optimized so that it increases the probability of selecting actions with higher critic values. Therefore, we combine our worst-attack critic and the original critic function, and optimize π_θ such that both the natural value ($\mathcal{L}_{\mathrm{RL}}$) and the worst-attack action value $\underline{Q}^{\pi}_{\phi}$ ($\mathcal{L}_{\mathrm{wst}}$) can be increased ($\mathcal{L}_{\mathrm{RL}}$ and $\mathcal{L}_{\mathrm{wst}}$). At the same time, π_θ is also regularized by $\mathcal{L}_{\mathrm{reg}}$.

We provide the full algorithm of WocaR-PPO in Algorithm 3 and highlight the differences with the prior method SA-PPO. WocaR-PPO needs to train an additional worst-attack critic $\underline{Q}^{\pi}_{\phi}$ to provide the robust-PPO-clip objective. The perturbation budget ϵ_t increases slowly during training. The implementation of $\mathcal{L}_{\mathrm{reg}}$ is the same as the SA-regularizer [19]. For computing the state importance weight w_{s_t}, because there is no Q-value network in PPO, we provide a different formula to measure the state importance without extra calculation (Line 11 in Algorithm 3).

3.7.2.3 Worst-case-aware Robust DQN (WocaR-DQN)

For value-based DRL methods [15, 79, 80] such as DQN, a Q network is learned to evaluate the natural action value. Although the policy is not directly modeled by a network, the Q network induces a greedy policy by $\pi(s) = \mathrm{argmax}_a Q(s,a)$. To distinguish the acting policy and the natural action value, we keep the original Q network, and learn a new Q network that serves as a robust policy. This new Q network is called a *robust Q network*, denoted by Q_r, which is used to take greedy actions $a = \pi(s) := \mathrm{argmax}_a Q_r(s,a)$ In addition to the original vanilla Q network Q_v and the robust Q network Q_r, we learn the worst-attack critic network $\underline{Q}^{\pi}_{\phi}$, which evaluates the worst-attack action value of the greedy policy induced by Q_r. Then, we update Q_r

Algorithm 3: Worst-case-aware Robust PPO (WocaR-PPO). We highlight the difference compares with SA-PPO [19] in blue.

Input: Number of iterations T, a schedule ϵ_t for the perturbation radius ϵ, weights $\kappa_{\text{wst}}, \kappa_{\text{reg}}$

Initialize policy network $\pi_{\theta_\pi}(a \mid s)$, value network $V_{\theta_V}(s)$ and worst-attack critic network $\underline{Q}^{\pi}_{\phi}(s,a)$ with parameters θ_π, θ_V and ϕ

for $k = 0, 1, ..., T$ **do**

Collect a set of trajectories $\mathcal{D} = \{\tau_k\}$ by running π_{θ_π} in the environment, each trajectory τ_k contains $\tau_k := \{(s_t, a_t, r_t, s_{t+1})\}, t \in [|\tau_k|]$

Compute rewards-to-go $\hat{R}_t$ for each step t in every trajectory k with discount factor γ

Compute advantage estimation $\hat{A}_t$ based on the current value function $V_{\theta_V}(s_t)$ and cumulative reward $\hat{R}_t$ for each step t

Update parameters of value function θ_V by regression on mean-squared error:

$$\theta_V \leftarrow \arg\min_{\theta_V} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} \left(V_{\theta_V}(s_t) - \hat{R}_t \right)^2.$$

// Use IBP to compute bounds of current policy network π

Find the upper bound $\overline{\pi}(s_{t+1}, \epsilon; \theta)$ and lower bound $\underline{\pi}(s_{t+1}, \epsilon; \theta)$ of the policy network π_{θ_π}

// Select the worst action for next states

Calculate the action satisfied $\hat{a}_{t+1} = \arg\min_{a \in [\underline{\pi}, \overline{\pi}]} \underline{Q}^{\pi}_{\phi}(s_{t+1}, a)$ with the worst-attack critic network $\underline{Q}^{\pi}_{\phi}$ using gradient descent.

// Compute next worst-case value:

Set $\underline{y}_t = \begin{cases} r_t & \text{for terminal } s_{t+1} \\ r_t + \gamma \underline{Q}^{\pi}_{\phi}(s_{t+1}, \hat{a}_{t+1}) & \text{for non-terminal } s_{t+1} \end{cases}$

Update parameters of worst-attack critic network ϕ by minimizing the TD-error ($\mathcal{L}_{\text{est}}$):

$$\phi \leftarrow \arg\min_{\phi} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} (\underline{y}_t - \underline{Q}^{\pi}_{\phi}(s_t, a_t))^2.$$

For each state s_t, calculate a state importance weight w_{s_t} by $V_{\theta_V}(s_t) - \min_a \underline{Q}^{\pi}_{\phi}(s_t, a)$ for s_t

Solve the value-enhanced state regularization loss by SGLD (Stochastic gradient Langevin dynamics [78]) (from SA-PPO [19]):

$$\mathcal{L}_{\text{reg}}(\pi_\theta) = \frac{1}{N} \sum_{t=1}^{N} w(s_t) \max_{\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)} \mathsf{Dist}(\pi_\theta(s_t), \pi_\theta(\tilde{s}_t)).$$

Update the policy network by minimizing the Robust-PPO-Clip objective (via ADAM):

$$\theta_\pi \leftarrow \arg\min_{\theta'_\pi} \frac{1}{|\mathcal{D}||\tau_k|} \left[\sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} \min \left(\rho_{\theta'_\pi}(a_t \mid s_t)(\hat{A}_t + \kappa_{\text{wst}} \underline{Q}^{\pi}_{\phi}(s_t, a_t)), g(\rho_{\theta'_\pi}(a_t \mid s_t)) \right)(\hat{A}_t + \kappa_{\text{wst}} \underline{Q}^{\pi}_{\phi}(s_t, a_t))) + \kappa_{\text{reg}} w(s_i) \mathcal{L}_{\text{reg}}(\pi_\theta) \right]$$

where $\rho_{\theta'_\pi}(a_t \mid s_t) := \frac{\pi_{\theta'_\pi}(a_t|s_t)}{\pi_{\theta_\pi}(a_t|s_t)}$, $g(\rho) := \text{clip}\left(\rho_{\theta'_\pi}(a_t \mid s_t), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}}\right)$

by assigning higher values for actions with both high natural Q value and high worst-attack action value ($\mathcal{L}_{\text{RL}}$ and $\mathcal{L}_{\text{wst}}$), while enforcing the network to output the same action under bounded state perturbations ($\mathcal{L}_{\text{reg}}$).

WocaR-DQN is presented in Algorithm 4. WocaR-DQN trains three Q-value functions including a vanilla Q network, a worst-case Q network, and a robust Q network. The worst-case Q $\underline{Q}^{\pi}_{\phi}$ is learned to estimate the worst-case performance and the robust Q is updated using the vanilla value and worst-case value together. Moreover, a target Q network is used as the original DQN implementation, to compute the target value when updating the vanilla Q network (Line 8 to 10 in Algorithm 4). To learn the worst-case critic $\underline{Q}^{\pi}_{\phi}$, we select the worst-attack action from the estimated possible perturbed action set $\hat{\mathcal{A}}_{\text{adv}}$ to compute the worst-case TD loss $\mathcal{L}_{\text{est}}$ (Line 11 to 15). The implementation of $\mathcal{L}_{\text{reg}}$ is the same as the SA-regularizer [19], where the robust Q network is regularized. To update the robust Q, we use a special y_i^r which combines the target Q $Q_{v'}$ and Q_r for the next state to compute the TD loss, and minimize the $\mathcal{L}_{\text{reg}}$ weighted by the state importance $w(s_i)$ (Line 16 to 17). In WocaR-DQN, we use an increasing ϵ_t schedule and a more slowly increasing worst-case schedule $\kappa_{wst}(t)$ for robust Q training.

3.7.3 Worst-case-aware Robust A2C (WocaR-A2C)

We also provide WocaR-A2C based on A2C implementation in Algorithm 5. Differ from the original A2C, WocaR-A2C needs to learn an additional $\underline{Q}^{\pi}_{\phi}$ similar to WocaR-PPO. To learn $\underline{Q}^{\pi}_{\phi}$, we compute the output bounds for the policy network π_{θ_π} under ϵ-bounded perturbations and then select the worst action $\hat{a}_{t+1}$ to calculate the TD-loss $\mathcal{L}_{\text{est}}$ (Line 6 to 9). The solutions for state importance weight $w(s_t)$ and regularization $\mathcal{L}_{\text{reg}}$ are same as WocaR-PPO (Line 10-11).

Algorithm 4: Worst-case-aware Robust DQN (WocaR-DQN). We highlight the difference compares with SA-DQN [19] in blue.

Input: Number of iterations T, target network update coefficient τ, a schedule ϵ_t for the perturbation radius ϵ, a worst-case schedule $\kappa_{\text{wst}}(t)$ for weight κ_{wst}, regularization weight κ_{reg}

Initialize a vanilla Q network $Q_v(s,a)$, target Q network $Q_{v'}(s,a)$, a robust Q network $Q_r(s,a)$, and a worst-attack critic $\underline{Q}^{\pi}_{\phi}(s,a)$ with parameters θ_{Q_v}, $\theta_{Q_{v'}}$, θ_{Q_r}, and ϕ

Initialize replay buffer $\mathcal{B}$

for $k = 0, 1, ..., T$ **do**

With probability β select random action a_t, otherwise select $a_t = \arg\max_a Q_r(s_t, a|\theta_{Q_r})$

Execute action a_t in environment and observe reward r_t and the next state s_{t+1}.

Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ in $\mathcal{B}$

Sample random a minibatch of N transitions $\{s_i, a_i, r_i, s_{i+1}\}$ from $\mathcal{B}$

Set $y_i = \begin{cases} r_i & \text{for terminal } s_{i+1} \\ r_i + \gamma \max_{a'} Q_{v'}(s_{i+1}, a'; \theta) & \text{for non-terminal } s_{i+1} \end{cases}$

Compute TD-loss for the vanilla Q network: $L(s_i, a_i, s_{i+1}; \theta) = (y_i - Q_v(s_i, a_i; \theta))^2$ and optimize θ_{Q_v}

Soft update the target action-value network: $\theta_{Q_{v'}} \leftarrow \tau\theta_{Q_v} + (1-\tau)\theta_{Q_{v'}}$

`// Computing bounds of robust action-value function:`

For each action a in action space $\mathcal{A}$, calculate the output bounds of robust action-value function Q_r under ϵ_t-bounded perturbations using IBP to input s_{i+1}: $Q_l(s_{t+1}, a, \epsilon_t)$ and $Q_u(s_{t+1}, a, \epsilon_t)$.

`// Find the possible perturbed action set:`

For every action $a \in \mathcal{A}$, if $Q_u(s_{t+1}, a, \epsilon_t) > Q_l(s_{t+1}, a', \epsilon_t), \forall a' \in \mathcal{A}$, then add a in the perturbed action set $\hat{\mathcal{A}}_{\text{adv}}$

Calculate the worst-attack action: $\hat{a}_{i+1} = \arg\min_{a \in \hat{\mathcal{A}}_{\text{adv}}} \underline{Q}^{\pi}_{\phi}(s_{i+1}, a)$.

Set $\underline{y}_i = \begin{cases} r_i & \text{for terminal } s_{i+1} \\ r_i + \gamma \underline{Q}^{\pi}_{\phi}(s_{i+1}, \hat{a}_{i+1}; \theta) & \text{for non-terminal } s_{i+1} \end{cases}$

Compute TD-loss for worst-attack critic: $\mathcal{L}_{\text{est}} = (\underline{y}_i - \underline{Q}^{\pi}_{\phi}(s_i, a_i; \phi))^2$ and perform a gradient descent step with respect to the parameters ϕ

Calculate the state importance w_{s_i} for each s_i by normalizing $\max_a Q_v(s_t, a) - \min_a Q_v(s_t, a)$

Update the robust Q function Q_r based on the modified TD-Loss and value-enhanced state regularization:

$$L(s_i, a_i, s_{i+1}; \theta_{Q_r}) = (y_i^r - Q_r(s_i, a_i; \theta))^2 + \kappa_{\text{reg}} w(s_i) \mathcal{L}_{\text{reg}}(\theta_{Q_r})$$

where $y_i^r = r_i + \gamma \max_{a'} \left[\kappa_{\text{wst}}(t) Q_{v'}(s_{i+1}, a'; \theta) + (1 - \kappa_{\text{wst}}(t)) \underline{Q}^{\pi}_{\phi}(s_{i+1}, a'; \theta) \right]$ if s_{i+1} is a non-terminal state, otherwise $y_i^r = r_i$

To learn the policy network π_{θ_π}, we minimize the $\underline{Q}^{\pi}_{\phi}$ value together with the original actor loss (Line 12).

Algorithm 5: Worst-case-aware Robust A2C (WocaR-A2C). We highlight the difference compares with SA-A2C [19] in blue.

Input: Number of iterations T, a schedule ϵ_t for the perturbation radius ϵ, weights $\kappa_{\text{wst}}, \kappa_{\text{reg}}$

Initialize policy network $\pi_{\theta_\pi}(a \mid s)$, value network $V_{\theta_V}(s)$ and worst-attack critic network $\underline{Q}^{\pi}_{\phi}(s,a)$ with parameters θ_π, θ_V and ϕ

for $k = 0, 1, ..., T$ **do**

Collect a set of trajectories $\mathcal{D} = \{\tau_k\}$ by running π_{θ_π} in the environment, each trajectory τ_k contains $\tau_k := \{(s_t, a_t, r_t, s_{t+1})\}, t \in [|\tau_k|]$

Compute advantage function A_t by $A_t = r_t + \gamma V_{\theta_V}(s_{t+1}) - V_{\theta_V}(s_t)$

Update parameters of value function θ_V by regression on mean-squared error:

$$\theta_V \leftarrow \arg\min_{\theta_V} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} A_t^2.$$

// Use IBP to compute bounds of current policy network π:

Find the upper bound $\overline{\pi}(s_{t+1}, \epsilon; \theta)$ and lower bound $\underline{\pi}(s_{t+1}, \epsilon; \theta)$ of the policy network π_{θ_π}

// Select the worst action for next states:

Calculate the action satisfied $\hat{a}_{t+1} = \arg\min_{a \in [\underline{\pi}, \overline{\pi}]} \underline{Q}^{\pi}_{\phi}(s_{t+1}, a)$ with the worst-attack critic network $\underline{Q}^{\pi}_{\phi}$ using gradient descent.

// Compute next worst-case value:

Set $\underline{y}_t = \begin{cases} r_t & \text{for terminal } s_{t+1} \\ r_t + \gamma \underline{Q}^{\pi}_{\phi}(s_{t+1}, \hat{a}_{t+1}) & \text{for non-terminal } s_{t+1} \end{cases}$

Update parameters of worst-attack critic network ϕ by minimizing the TD-error ($\mathcal{L}_{\text{est}}$):

$$\phi \leftarrow \arg\min_{\phi} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} (\underline{y}_t - \underline{Q}^{\pi}_{\phi}(s_t, a_t))^2.$$

For each state s_t, calculate a state importance weight $w(s_t)$ by $V_{\theta_V}(s_t) - \min_a \underline{Q}^{\pi}_{\phi}(s_t, a)$ for s_t

Solve the value-enhanced state regularization loss [72] by SGLD (Stochastic gradient Langevin dynamics [78]):

$$\mathcal{L}_{\text{reg}}(\pi_{\theta_\pi}) = \frac{1}{N} \sum_{t=1}^{N} w(s_t) \max_{\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)} \mathsf{Dist}(\pi_{\theta_\pi}(s_t), \pi_{\theta_\pi}(\tilde{s}_t)).$$

Update the policy network by (via ADAM)

$$\theta_\pi \leftarrow \arg\min_{\theta'_\pi} \frac{1}{|\mathcal{D}||\tau_k|} \left[\sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} (A_t log \pi_{\theta_\pi}(s_t) + \kappa_{\text{wst}} \underline{Q}^{\pi}_{\phi}(s_t, a_t)) \right]$$

3.7.3.1 Extension to Action Attacks

Although our paper mainly focuses on state attack, our proposed techniques and algorithms based on the worst-attack Bellman operator can be easily extended to action attack, which is another threat model studied in previous works [28, 29, 36]. In fact, for action attack, we even do not need to apply IBP for the worst-attack Bellman backup. We could just simply replace $\mathcal{A}_{\text{adv}}$ with the set of actions that the agent could take under attack, then the rest of the algorithms will follow the exact same as the ones presented here.

3.7.4 Experiment Details and Additional Results

3.7.4.1 Implementation Details

For reproducibility, the reported results are selected from 30 agents for different training methods with medium performance due to the high variance in RL training.

A. PPO in MuJoCo

(a) PPO Baselines

Vanilla PPO We use the optimal hyperparameters from [19] with the original fully connected (MLP) structure as the policy network for vanilla PPO training on all environments. On Hopper, Walker2d and Halfcheetah, we train for 2 million steps (976 iterations) , and 10 million steps (4882 iterations) on Ant to ensure convergence, which are consistent with other baselines (except ATLA methods).

SA-PPO We use the hyperparameters using a grid search and solve the regularizer using convex relaxation with the IBP+Backward scheme to solve the regularizer. The regularization parameter

$kappa$ is chosen in $\{0.01, 0.03, 0.1, 0.3, 1.0\}$.

ATLA-PPO The hyperparameters for both policy and adversary are tuned for vanilla PPO with LSTM models. A larger entropy bonus coefficient is set to allow sufficient exploration. We set $N_v = N_\pi = 1$ for all experiments. We train 2441 iterations for Hopper, Walker2d, and Halfcheetah as well as 4882 iterations for Ant.

PA-ATLA-PPO We use the hyperparameters similar to ATLA-PPO and conduct a grid search for a part of adversary hyperparameters including the learning rate and the entropy bonus coefficient.

RADIAL-PPO RADIAL-PPO applies the same value of hyperparameters from [39]. We train agents with the same iterations aligning vanilla PPO for fair comparison.

(b) PPO Attackers

For **Random** and **MaxDiff** attack, we directly use the implementation from [18]. The reported rewards under RS attack are from 30 trained robust value function, which is used to attack agents.

For **SA-RL** attack, a grid search of the optimal hyperparameters for each robust agents is conducted to find the strongest attacker. The strength of the regularization κ is set as 1×10^{-6} to 1.

For **PA-AD** attack, the adversaries are trained by PPO with a grid search of hyperparameters to obtain the strongest adversary.

For different types of RL-based attacks, we respectively train 100 adversaries and report the worst rewards among all trained adversaries.

(c) WocaR-PPO We use the same LSTM structure (single layer with 64 hidden neurons as in vanilla PPO agents. With a grid search experiment, we find the optimal hyperparameters for WocaR-PPO. Specially, we use PGD to compute bounds for the policy network and convex

relaxation to solve the state regularization. The number of WocaR-PPO training steps in all environments are the same as those in vanilla PPO. We tune the adjustable weight κ_{wst} and increase κ_{wst} from 0 to the target value. For Hopper, Walker2d and Halfcheetah, κ_{wst} is linearly increasing and we set the target value as 0.8. For Ant, we choose the exponential increase and the target value as 0.5.

B. DQN in Atari

(a) DQN Baselines

Vanilla DQN We follow [19] and [39] in hyperparameters and network structures for vanilla DQN training. The implementation of all our baselines applies Double DQN [81] and Prioritized Experience Replay [82]. For each Atari environment without framestack, we normalize the pixel values to $[0, 1]$ and clip rewards to $[-1, +1]$. For reliably convergence, we run 6×10^6 steps for all baselines on all environments. Additionally, we use a replay buffer with a capacity of 5×10. During testing, we evaluate agents without epsilon greedy exploration for 1000 episodes.

SA-DQN SA-DQN use the same settings of network structures and hyperparameters as in vanilla DQN. The regularization parameter κ is chosen from $0.005, 0.01, 0.02$ and the schedule of ϵ during training also follows [19].

RADIAL-DQN Following the original implementation from [39], we reproduce the results of RADIAL-DQN with our environment settings.

(b) DQN Attackers

For **PGD** attacks, we apply 10-step untargeted PGD attacks. We also try 50-step PGD attacks, but we find that the rewards of robust agents do not further reduce.

For **MinBest** attacks, we use FGSM to compute state perturbations following [4].

For **PA-AD** attacks, the PA-AD attackers are learned with the ACKTR algorithm. We use a

learning rate 0.0001 and train the attackers for 5 million frames.

(c) WocaR-DQN For WocaR-DQN, we keep the same network architectures and hyper-parameters as in vanilla DQN agents. During training, we set the adjustable weight κ_{wst} as 0 for the first 2×10^6 steps, and then exponentially increase it from 0 to 0.5 for 4×10^6 steps.

3.7.4.2 Additional Experiment Results on Robustness Performance

Environment	Model	Natural Reward	Random	MAD	RS	SA-RL	PA-AD
Halfcheetah state-dim: 17 ϵ=0.15	PPO (vanilla)	**7117 ± 98**	5486 ± 1378	1836 ± 866	489 ± 758	-660 ± 218	-356 ± 407
	SA-PPO	3632 ± 20	3619 ± 18	3624 ± 23	3283 ± 20	3028 ± 23	2512 ± 16
	ATLA-PPO	6157 ± 852	6164 ± 603	5790 ± 174	4806 ± 392	5058 ± 418	2576 ± 548
	PA-ATLA-PPO	6289 ± 342	**6215 ± 346**	**5961 ± 253**	5226 ± 114	4872 ± 379	3840 ± 273
	RADIAL-PPO	4724 ± 14	4731 ± 42	3994 ± 156	3864 ± 232	3253 ± 131	2674 ± 168
	WocaR-PPO (Ours)	6032 ± 68	5969 ± 149	5850 ± 228	**5319 ± 220**	**5365 ± 54**	**4269 ± 172**
Hopper state-dim: 11 ϵ=0.075	PPO (vanilla)	3167 ± 542	2101 ± 793	1410 ± 655	794 ± 238	636 ± 9	160 ± 136
	SA-PPO	3705 ± 2	2710 ± 801	2652 ± 835	1130 ± 42	1076 ± 791	856 ± 21
	ATLA-PPO	3291 ± 600	3165 ± 576	2814 ± 725	2244 ± 618	1772 ± 802	1232 ± 350
	PA-ATLA-PPO	3449 ± 237	3325 ± 239	3145 ± 546	3002 ± 329	1529 ± 284	2521 ± 325
	RADIAL-PPO	**3740 ± 44**	**3729 ± 100**	3214 ± 142	2141 ± 232	1722 ± 186	1439 ± 204
	WocaR-PPO (Ours)	3616 ± 99	3633 ± 30	**3541 ± 207**	**3277 ± 159**	**2390 ± 145**	**2579 ± 229**
Walker2d state-dim: 17 ϵ=0.05	PPO (vanilla)	4472 ± 635	3007 ± 1200	2869 ± 1271	1336 ± 654	1086 ± 516	804 ± 130
	SA-PPO	4487 ± 61	4465 ± 39	3668 ± 689	3808 ± 138	2908 ± 336	1042 ± 353
	ATLA-PPO	3842 ± 475	3927 ± 368	3836 ± 492	3239 ± 294	3663 ± 707	1224 ± 770
	PA-ATLA-PPO	4178 ± 529	4129 ± 78	4024 ± 272	3966 ± 307	3450 ± 178	2248 ± 131
	RADIAL-PPO	**5251 ± 12**	**5184 ± 42**	**4494 ± 150**	3572 ± 239	3320 ± 245	1395 ± 194
	WocaR-PPO (Ours)	4156 ± 495	4244 ± 157	4177 ± 176	**4093 ± 138**	**3770 ± 196**	**2722 ± 173**
Ant state-dim: 111 ϵ=0.15	PPO (vanilla)	**5687 ± 758**	5261 ± 1005	1759 ± 828	268 ± 227	-872 ± 436	-2580 ± 872
	SA-PPO	4292 ± 384	4986 ± 452	4662 ± 522	3412 ± 1755	2511 ± 1117	-1296 ± 923
	ATLA-PPO	5359 ± 153	5366 ± 104	5240 ± 170	4136 ± 149	3765 ± 101	220 ± 338
	PA-ATLA-PPO	5469 ± 106	5496 ± 158	**5328 ± 196**	4124 ± 291	3694 ± 188	2986 ± 364
	RADIAL-PPO	5076 ± 254	5031 ± 142	4777 ± 156	3731 ± 177	3188 ± 115	1544 ± 194
	WocaR-PPO (Ours)	5596 ± 225	**5558 ± 241**	5284 ± 182	**4339 ± 160**	**3822 ± 185**	**3164 ± 163**

Table 3.2: Average episode rewards ± standard deviation over 50 episodes on five baselines and WocaR-PPO on Hopper, Walker2d, Halfcheetah, and Ant. Natural reward and rewards under five types of attacks are reported. Under each column corresponding to an evaluation metric, we bold the best results. And the row for the most robust agent is highlighted as gray. Note that *ATLA-PPO, PA-ATLA-PPO and RADIAL-PPO are trained with more than* $2\times$ *steps than WocaR-PPO*, as reported in Table 3.6.

MuJoCo Experiments We reported all results in Table 3.2 including episode rewards of well-trained robust models under various adversarial attacks. Under this full adversarial evaluation, we provide a robustness comparison between baselines and our algorithm from a comprehensive angle. We report the attack performance under a common chosen perturbation budget ϵ following [18, 19]. Results in all four MuJoCo environments show that our WocaR-PPO is the most robust method. We emphasize that Table 3.2 reports the final performance of all robust training baselines after convergence, but some baselines takes much more steps than our WocaR-PPO. Table 3.5 in Section 3.7.4.4 compares all methods under the same number of training steps, where WocaR-PPO outperforms baselines more significantly.

Atari Experiments In Table 3.3, we present performance based on DQN on four Atari environments under 1/255 and 3/255 ϵ attack. Under ϵ of 1/255, our WocaR-DQN achieves competitive performance under PGD attacks and outperforms all baselines under MinBest and PA-AD attacks, which shows better robustness of WocaR-DQN under weaker attacks.

Based on vanilla A2C, we implement SA-A2C[19] and PA-ATLA-A2C[8] as robust baselines. We implement WocaR-A2C to compare with ATLA methods on Atari. In Table 3.4, under any ϵ value, our WocaR-A2C outperforms other robust baselines across different attacks. We can conclude that our method considerably enhance more robustness than ATLA methods on Atari.

3.7.4.3 Robustness Evaluation Using Multiple ϵ

To study how WocaR-PPO performs under attacks with different value of ϵ, Figure 3.9 shows the evaluation of our algorithms under different ϵ attacks compared with the baselines in Hopper and Walker2d. We can conclude that our robustly trained model universally and signifi-

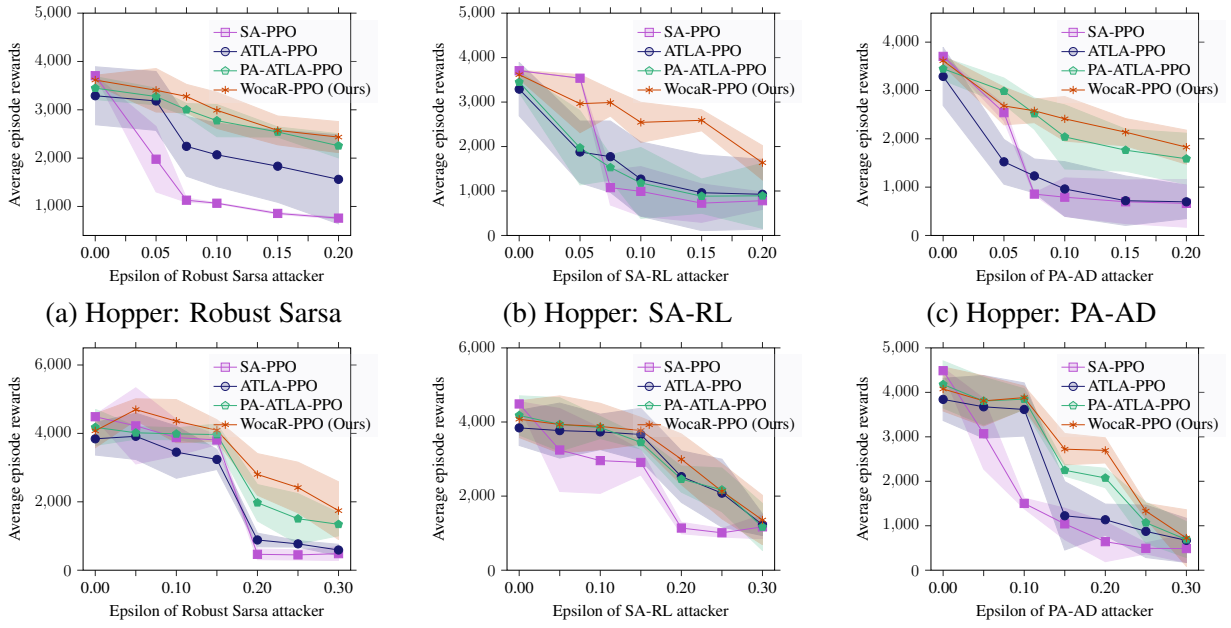
Environment	Model	Natural Reward	PGD (10 steps)		MinBest		PA-AD	
			ϵ=1/255	ϵ=3/255	ϵ=1/255	ϵ=3/255	ϵ=1/255	ϵ=3/255
Pong	DQN	21.0 ± 0.0	-21.0 ± 0.0	-21.0 ± 0.0	-7.4 ± 2.8	-9.7 ± 4.0	-18.2 ± 2.3	-19.0 ± 2.2
	SA-DQN	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	20.6 ± 3.5	20.4 ± 1.8	18.7 ± 2.6
	RADIAL-DQN	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	19.5 ± 2.1	20.3 ± 2.5	13.2 ± 1.8
	WocaR-DQN (Ours)	**21.0 ± 0.0**	**21.0 ± 0.0**	**21.0 ± 0.0**	**21.0 ± 0.0**	**20.8 ± 3.3**	**21.0 ± 0.2**	**19.7 ± 2.4**
Freeway	DQN	**34.0 ± 0.1**	0.0 ± 0.0	0.0 ± 0.0	9.5 ± 3.0	5.5 ± 1.8	9.3 ± 2.7	4.7 ± 2.9
	SA-DQN	30.0 ± 0.0	30.0 ± 0.0	30.0 ± 0.0	27.2 ± 3.4	18.3 ± 3.0	20.1 ± 4.0	9.5 ± 3.8
	RADIAL-DQN	33.1 ± 0.2	**33.1 ± 0.2**	**33.2 ± 0.2**	22.6 ± 3.3	16.4 ± 2.3	18.5 ± 4.2	10.8 ± 3.6
	WocaR-DQN (Ours)	31.2 ± 0.4	31.2 ± 0.5	31.4 ± 0.3	**29.6 ± 2.5**	**19.8 ± 3.8**	**24.9 ± 3.7**	**12.3 ± 3.2**
BankHeist	DQN	**1308 ± 24**	54 ± 20	0 ± 0	210 ± 79	119 ± 65	213 ± 111	102 ± 92
	SA-DQN	1245 ± 14	**1245 ± 10**	1176 ± 63	1148 ± 36	1024 ± 31	1054 ± 11	489 ± 106
	RADIAL-DQN	1178 ± 4	1178 ± 4	1176 ± 63	1049 ± 27	928 ± 113	1035 ± 46	508 ± 85
	WocaR-DQN (Ours)	1220 ± 12	1220 ± 3	**1214 ± 7**	**1192 ± 12**	**1045 ± 20**	**1096 ± 19**	**754 ± 102**
RoadRunner	DQN	**45527 ± 4894**	0 ± 0	0 ± 0	14962 ± 6431	2985 ± 1440	842 ± 41	203 ± 65
	SA-DQN	44638 ± 2367	43970 ± 975	20678 ± 1563	39736 ± 2315	4214 ± 2587	38432 ± 3574	5516 ± 4684
	RADIAL-DQN	44675 ± 5854	**44605 ± 1094**	38576 ± 1960	38060 ± 1799	8476 ± 3964	36310 ± 9149	1290 ± 4015
	WocaR-DQN (Ours)	44156 ± 2279	44079 ± 2154	**38720 ± 1765**	**40758 ± 3369**	**10545 ± 2984**	**38954 ± 3647**	**8239 ± 2766**

Table 3.3: Average episode rewards ± standard deviation over 1000 episodes on baselines and WocaR-DQN on Pong, Freeway, BankHeist, and RoadRunner. Natural reward and rewards under different attacks with ϵ of 1/255 and 3/255 are reported. We bold the best results for each evaluation metric. And the row for the most robust agents on all environments are highlighted by gray.

Environment	Model	Natural Reward	PGD (10 steps)		MinBest		PA-AD	
			ϵ=1/255	ϵ=3/255	ϵ=1/255	ϵ=3/255	ϵ=1/255	ϵ=3/255
BankHeist	A2C	**1228 ± 93**	67 ± 14	0 ± 0	972 ± 99	697 ± 153	636 ± 74	314 ± 116
	SA-A2C	1029 ± 152	1029 ± 156	976 ± 54	902 ± 89	786 ± 52	836 ± 70	644 ± 153
	PA-ATLA-A2C	1076 ± 56	1075 ± 79	1013 ± 69	957 ± 78	842 ± 154	862 ± 106	757 ± 132
	WocaR-A2C (Ours)	1089 ± 34	**1089 ± 78**	**1035 ± 102**	**1043 ± 29**	**937 ± 65**	**1004 ± 94**	**879 ± 128**

Table 3.4: Average episode rewards ± standard deviation over 1000 episodes on baselines and WocaR-A2C on BankHeist. Natural reward and rewards under different attacks with ϵ of 1/255 and 3/255 are reported. We bold the best results for each evaluation metric. And the row for the most robust agents on all environments are highlighted by gray.

cantly outperforms other robust agents considering various attack budget ϵ.



(a) Hopper: Robust Sarsa (b) Hopper: SA-RL (c) Hopper: PA-AD

(d) Walker2d: Robust Sarsa (e) Walker2d: SA-RL (f) Walker2d: PA-AD

Figure 3.9: Comparisons under different attacks w.r.t. different budget ϵ's on Hopper and Walker2d.

Environment	Model	Natural Reward	Random	MAD	RS	SA-RL	PA-AD
Halfcheetah state-dim: 17 ϵ=0.15	ATLA-PPO	4817 ± 277	4809 ± 186	4584 ± 100	4074 ± 285	4129 ± 348	1856 ± 294
	PA-ATLA-PPO	5023 ± 282	5076 ± 149	4720 ± 334	4392 ± 158	4159 ± 248	3085 ± 295
	RADIAL-PPO	4683 ± 97	4625 ± 190	3674 ± 222	3529 ± 173	2893 ± 165	2197 ± 251
	WocaR-PPO (Ours)	**6032 ± 68**	**5969 ± 149**	**5850 ± 228**	**5319 ± 220**	**5365 ± 54**	**4269 ± 172**
Hopper state-dim: 11 ϵ=0.075	ATLA-PPO	3265 ± 342	3195 ± 275	2675 ± 332	2098 ± 398	1542 ± 639	1135 ± 289
	PA-ATLA-PPO	3429 ± 196	3455 ± 315	3072 ± 478	2889 ± 258	1458 ± 274	2032 ± 244
	RADIAL-PPO	**3687 ± 80**	3627 ± 106	2952 ± 126	1094 ± 248	1243 ± 187	1036 ± 142
	WocaR-PPO (Ours)	3616 ± 99	**3633 ± 30**	**3541 ± 207**	**3277 ± 159**	**2390 ± 145**	**2579 ± 229**
Walker2d state-dim: 17 ϵ=0.05	ATLA-PPO	2664 ± 366	2695 ± 320	2547 ± 210	2439 ± 174	2092 ± 144	1544 ± 280
	PA-ATLA-PPO	3047 ± 223	3112 ± 111	2865 ± 230	2742 ± 177	2450 ± 229	1987 ± 246
	RADIAL-PPO	2143 ± 153	2231 ± 89	2095 ± 121	1680 ± 193	1078 ± 115	1274 ± 117
	WocaR-PPO (Ours)	**4156 ± 495**	**4244 ± 157**	**4177 ± 176**	**4093 ± 138**	**3770 ± 196**	**2722 ± 173**
Ant state-dim: 111 ϵ=0.15	ATLA-PPO	4249 ± 243	4218 ± 161	4036 ± 173	3391 ± 158	2045 ± 203	-349 ± 175
	PA-ATLA-PPO	4533 ± 238	4492 ± 190	4232 ± 203	3579 ± 261	2762 ± 152	1765 ± 185
	RADIAL-PPO	4379 ± 230	4194 ± 52	3278 ± 138	2348 ± 232	1380 ± 145	157 ± 124
	WocaR-PPO (Ours)	**5596 ± 225**	**5558 ± 241**	**5284 ± 182**	**4339 ± 160**	**3822 ± 185**	**3164 ± 163**

Table 3.5: Average episode rewards ± standard deviation over 50 episodes on baselines and WocaR-PPO trained for 2 million steps on Hopper, Walker2d, Halfcheetah and 7.5 million steps on Ant (less than the best settings). **Bold** numbers indicate the best results under each attack. The gray rows are the most robust agents.

Model	Hopper		Ant	
	Time (h)	Steps(m)	Time (h)	Steps (m)
SA-PPO	3.0	2.0	8.9	10.0
ATLA-PPO	5.6	5.0	12.8	10.0
PA-ATLA-PPO	5.2	5.0	12.3	10.0
RADIAL-PPO	3.2	4.0	10.2	10.0
WocaR-PPO (Ours)	**2.3**	**2.0**	**8.7**	**7.5**

Table 3.6: Efficiency comparison of state-of-the-art robust training methods and WocaR-PPO in Hopper and Ant. For Walker2d and Halfcheetah, the sampling steps are same as for Hopper and the training time is also extremely similar. We highlight the most efficient method as gray.

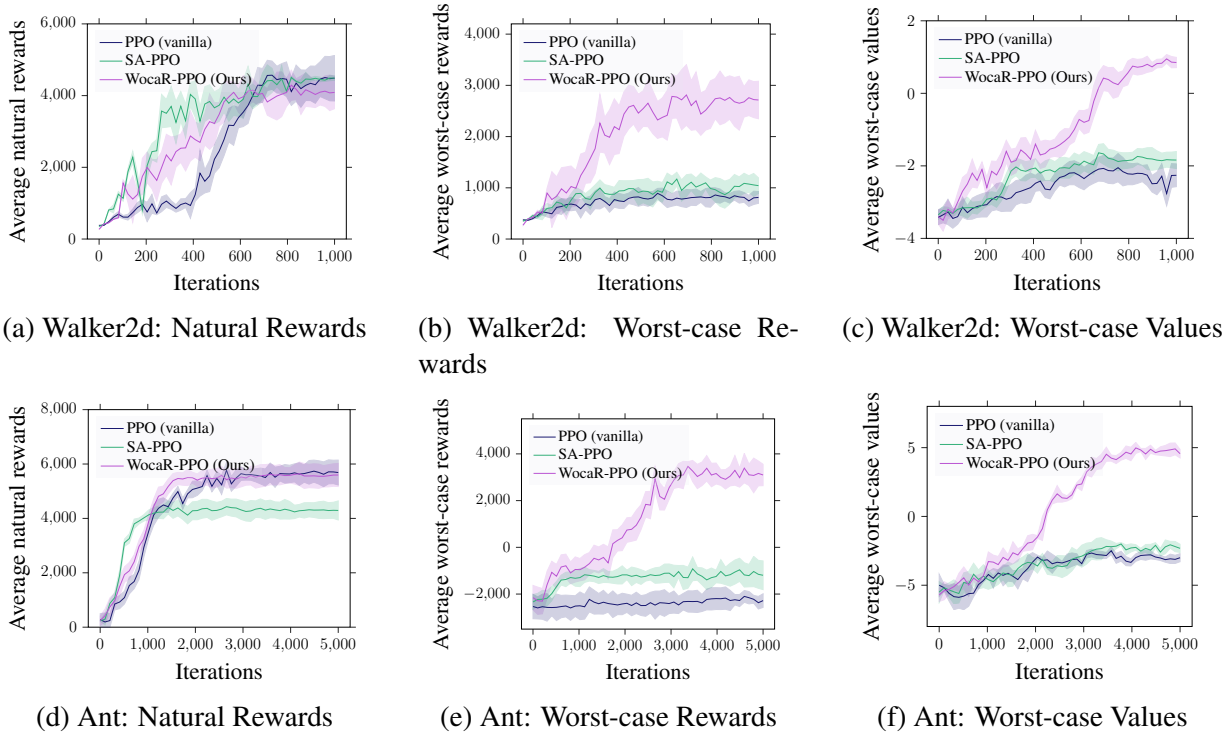
3.7.4.4 Additional Evaluation on Sample Efficiency

In Table 3.5, we report the performance of WocaR-PPO and all robust PPO baselines using the same training steps. We find that under limited training steps, ATLA-PPO, PA-ATLA-PPO and RADIAL-PPO obtain sub-optimal robustness, which suggests that these methods are more sample-hungry. In contrast, WocaR-PPO converges under fewer steps and achieves best performance with a large advantage, which shows the higher efficiency of WocaR-PPO.

3.7.4.5 Additional Results of Time Efficiency

We show the training efficiency of WocaR-PPO from three aspects including time, training iterations, and sampling in MuJoCo environments by comparing with SA-PPO and state-of-the-art methods ATLA-PPO, PA-ATLA-PPO, and RADIAL-PPO in Table 3.6. For a fair comparison, we use the same *GeForce RTX 1080 Ti GPUs* to train all the robust agents.

It needs to mention that in continuous action spaces when estimating the worst-case value, we solve $\min_{\hat{a}\in\hat{\mathcal{A}}_{\text{adv}}} \underline{Q}^{\pi}_{\phi}(s_{t+1},\hat{a})$ using 50-step gradient descent. The running time of this 50-step gradient descent is about **1.68 seconds** per batch with batch size 128. In total, this gradient



(a) Walker2d: Natural Rewards (b) Walker2d: Worst-case Rewards (c) Walker2d: Worst-case Values

(d) Ant: Natural Rewards (e) Ant: Worst-case Rewards (f) Ant: Worst-case Values

Figure 3.10: Learning curves (mean $\pm$ standard deviation) of natural rewards, worst-case rewards under attacks and estimated worst-case values during training on Walker2d and Ant for vanilla PPO (blue), SA-PPO (green) and WocaR-PPO (purple).

descent computation takes 18% of the total training time, thus it is not the computation bottleneck.

Without training with an adversary, *our algorithm requires much less (only 50% or 75%) steps to reliably converge.* WocaR-PPO only takes less than half of time for low-dimensional environments to converge compared to ATLA methods and RADIAL-PPO. In high-dimensional environments like Ant, we only need 4 hours for training, while ATLA methods require at least 7 hours. When solving harder tasks, the efficiency advantage of WocaR-PPO is more obvious.

3.7.4.6 Effectiveness of Worst-attack Policy Optimization

In addition to Figure 3.7, we show the learning curves in Walker2d and Ant in Figure 3.10 to verify the effectiveness of worst-attack value estimation and worst-case-aware policy

optimization. Figure 3.10(a) and (d) show the natural rewards of agents during training without attacks. The actual worst-attack rewards in Figure 3.10(b) and (e) refer to the the reward obtained by the agents under PA-AD attack [8] which is the existing strongest attacking algorithm. To study the worst-case performance during training, We evaluate PPO, SA-PPO and WocaR-PPO agents after every 20 iterations using all types of attacks and report the worst-case rewards for each checkpoint. We also present the trend of the estimated worst-case values during training in Figure 3.10(c) and (f), which are tested by the trained worst-attack value functions $\underline{Q}^{\pi}_{\phi}$.

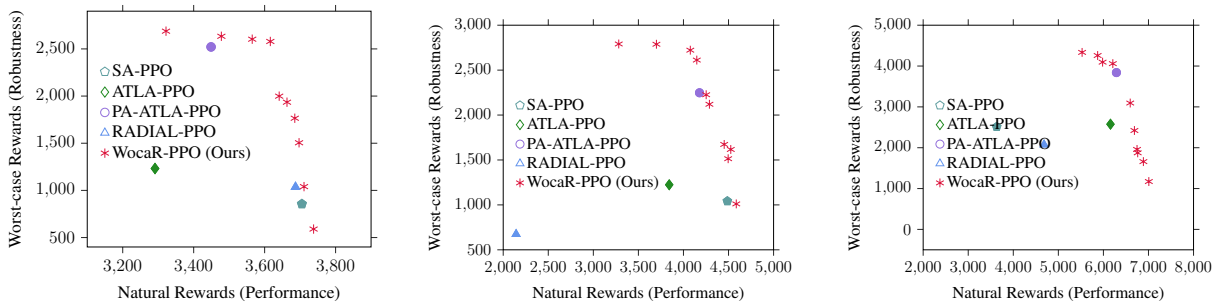
We observe from the curves that our worst-attack critic estimation matches the trend of actual worst-attack rewards. Also, the increases of estimated worst-attack values and actual worst-attack rewards of WocaR-PPO show that our WocaR-RL significantly improves the robustness of agents by enhancing worst-attack values.

3.7.4.7 Trade-off between Natural Performance and Robustness

As mentioned in Section 3.5.2, the adjustable weight κ_{wst} controls the trade-off between natural performance and robustness. To discuss the effect of κ_{wst}, we train agents using WocaR-PPO in Hopper, Walker2d, and Halfcheetah with uniformly sampled 40 different values of weight κ_{wst} in range $(0, 1]$.

Figure 3.11 plots the worst-case performance and natural performance of robust training baselines and 10 agents trained by WocaR-PPO with various values of κ_{wst}. We can see that when reward under worst-case perturbations increases, it leads to a reduction of the natural reward.

The choice of the worst-case value's weight κ_{wst} is to control the trade-off between the final natural performance and robustness. It does not affect the convergence of the algorithm. When



(a) Hopper (b) Walker2d (c) Halfcheetah

Figure 3.11: Average natural rewards and worst-case rewards of WocaR-PPO with different κ_{wst} and other baselines on Hopper, Walker2d, and Halfcheetah.

we increase the weight of worst-case values κ_{wst}, the reward under worst-case perturbations increases, but it leads to a reduction of the natural reward. Equally, when κ_{wst} is set close to 0, the algorithm is similar to standard training, where the policy achieves high reward under no attack, but extremely low reward under attacks. Hence, κ_{wst} is necessary for our algorithm to balance these two kinds of performance. In practice, one can adjust κ_{wst} according to their preferences to robustness and natural performance.

We report the results in Table 3.2 with significant better worst-case robustness and comparable natural performance compared with baselines. WocaR-PPO can always find policies which dominate other robust agents.

3.7.4.8 Additional Ablation Studies

We provide full ablation experimental results for the state importance weight $w(s)$ and the regularization loss $\mathcal{L}_{\text{reg}}$ [19] on four MuJoCo environments.

For the state importance weight $w(s)$, we compare the performance between the original WocaR-PPO and WocaR-PPO without $w(s)$ in Figure 3.12. Additionally, we also equip SA-PPO with $w(s)$ to show the universal applicability of this design. In all four MuJoCo environments,

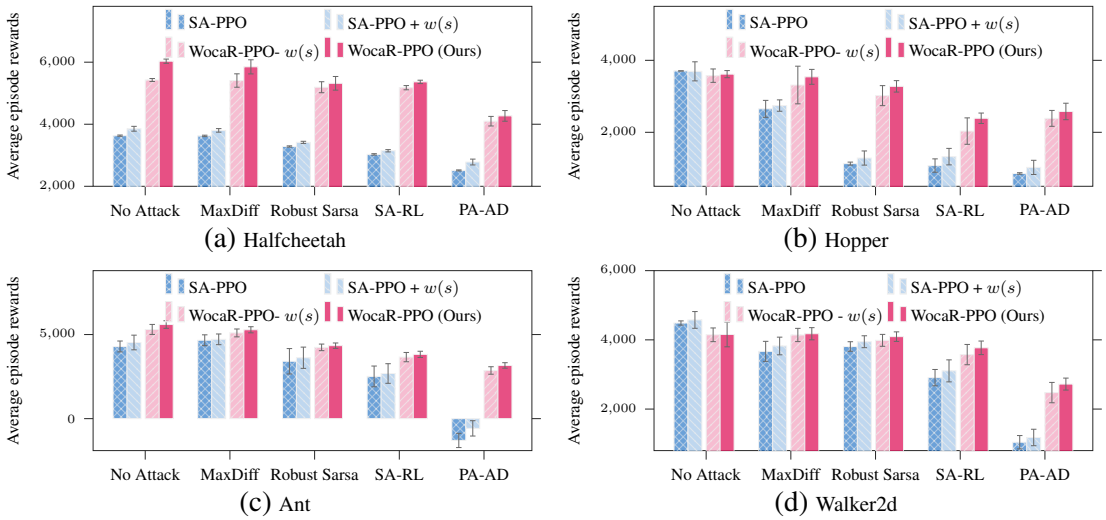


Figure 3.12: Ablation performance for the state importance weight $w(s)$ under no attack and different attacks on Hopper, Walker2d, Halfcheetah, and Ant.

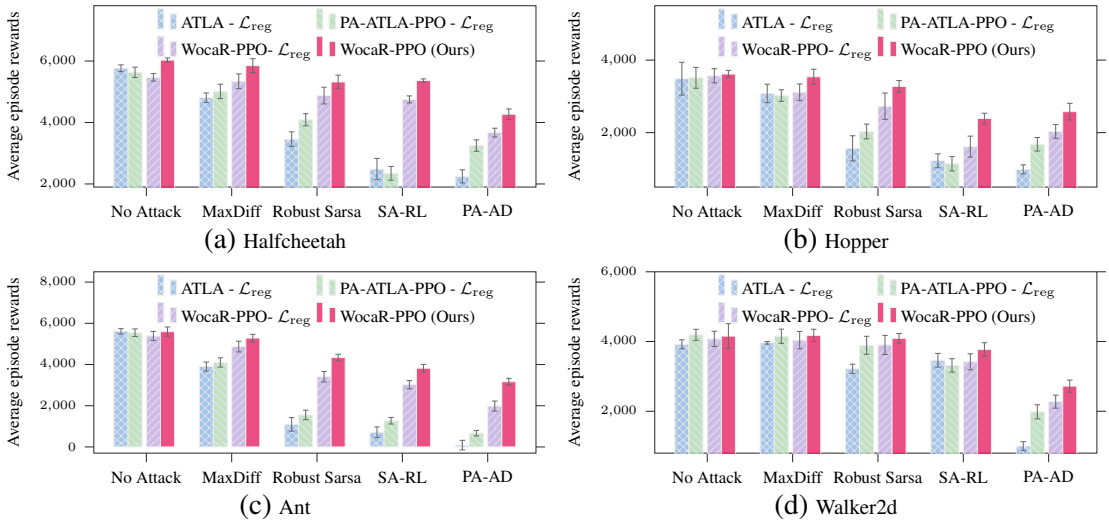


Figure 3.13: Ablation performance for the state regularization loss $\mathcal{L}_{\text{reg}}$ under no attack and different attacks on Hopper, Walker2d, Halfcheetah, and Ant.

we can see that with $w(s)$, both WocaR-PPO and SA-PPO get boosted robustness, verifying the effectiveness of the state importance weight.

For the state regularization loss $\mathcal{L}_{\text{reg}}$, Figure 3.13 verifies that $\mathcal{L}_{\text{reg}}$ enhances the robustness of WocaR-PPO, since the performance of WocaR-PPO drops without $\mathcal{L}_{\text{reg}}$. On the other hand, Figure 3.13 also compares the performance of ATLA methods and our algorithm without $\mathcal{L}_{\text{reg}}$

(note that ATLA methods also regularizes the PPO policies during training). The results indicate that *the decisive contribution of WocaR-PPO to robustness improving comes from the worst-attack-aware policy optimization.*

These ablation studies demonstrate that all the techniques are beneficial for robustness improvement and further show that our worst-case-aware training performs better than training with attackers.

Chapter 4: Robustness of RL under Training-Time Poisoning

4.1 Introduction

Although reinforcement learning (RL), especially deep RL, has been successfully applied in various fields, the security of RL techniques against adversarial attacks is not yet well understood. In real-world scenarios, including high-stakes ones such as autonomous driving vehicles and healthcare systems, a bad decision may lead to a tragic outcome. Should we trust the decision made by an RL agent? How easy is it for an adversary to mislead the agent? These questions are crucial to ask before deploying RL techniques in many applications.

In this chapter, we focus on *poisoning attacks*, which occur during the training and influence the learned policy. Since training RL is known to be very sample-consuming, one might have to constantly interact with the environment to collect data, which opens up a lot of opportunities for an attacker to poison the training samples collected. Therefore, understanding poisoning mechanisms and studying the vulnerabilities in RL are crucial to provide guidance for defense methods. However, existing works on adversarial attacks in RL mainly study the test-time *evasion attacks* [83] where the attacker crafts adversarial inputs to fool a well-trained policy, but does not cause any change to the policy itself. Motivated by the importance of understanding RL security in the training process and the scarcity of relevant literature, in this chapter, we *investigate how to poison RL agents and how to characterize the vulnerability of deep RL algorithms.*

In general, RL is an "online" process: an agent rolls out experience from the environment with its current policy, and uses the experience to improve its policy, then uses the new policy to roll out new experience, etc. Poisoning in online RL is significantly different from poisoning in classic supervised learning (SL), even online SL, and is more difficult due to the following challenges.

Challenge I – Future Data Unavailable in Online RL. Poisoning approaches in SL [84, 85] usually require the access to the whole training dataset, so the attacker can decide the optimal poisoning strategy before the learning starts. However, in online RL, the training data (trajectories) are generated by the agent while it is learning. Although the optimal poison should work in the long run, the attacker can only access and change the data in the current iteration, since the future data is not yet generated.

Challenge II – Data Samples No Longer i.i.d.. It is well-known that in RL, data samples (state-action transitions) are no longer i.i.d., which makes learning challenging, since we should consider the long-term reward rather than the immediate result. However, we notice that data samples being not i.i.d. also makes poisoning attacks challenging. For example, an attacker wants to reduce the agent's total reward in a task shown as Figure 4.1; at state s_1, the attacker finds that a_1 is less rewarding than a_0; if the attacker only looks at the immediate reward, he will lure the agent into choosing a_1. However, following a_1 finally leads the agent to s_{10} which has a much higher reward.

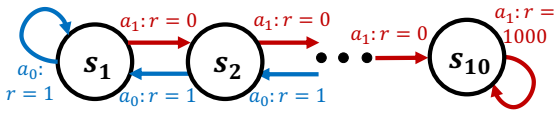


Figure 4.1: An example of difficult poisoning.

Challenge III – Unknown Dynamics of Environment. Although Challenge I and II can be partially addressed by predicting the future trajectories or steps, it requires prior knowledge on

the dynamics of the underlying MDP. Many existing poisoning RL works [35, 86] assume the attacker has perfect knowledge of the MDP, then compute the optimal poisoning. However, in many real-world environments, knowing the dynamics of the MDP is difficult. Although the attacker could potentially interact with the environment to build an estimate of the environment model, the cost of interacting with the environment could be unrealistically high, market making [87] for instance. In this chapter, we study a more realistic scenario where the attacker does not know the underlying dynamics of MDP, and can not directly interact with the environment, either. Thus, the attacker learns the environment only based on the agent's experience.

In this chapter, we systematically investigate poisoning in RL by considering all the aforementioned RL-specific challenges. Previous works either do not address any of the challenges or only address some of them. Behzadan et al. [31] achieve policy induction attacks for deep Q networks (DQN). However, they treat output actions of DQN similarly to labels in SL, and do not consider Challenge II that the current action will influence future interactions. Ma et al. [86] propose a poisoning attack for model-based RL, but they suppose the agent learns from a batch of given data, not considering Challenge I. Rakhsha et al. [35] study poisoning for online RL, but they require perfect knowledge of the MDP dynamics, which is unrealistic as stated in Challenge III.

Summary of Contributions.

(1) We propose a practical poisoning algorithm called Vulnerability-Aware Adversarial Critic Poison (VA2C-P) that works for deep policy gradient learners without any prior knowledge of the environment. To the best of our knowledge, VA2C-P is *the first practical algorithm that poisons policy-based deep RL methods*.

(2) We introduce a novel metric, called stability radius, to characterize the stability of RL algo-

rithms, measuring and comparing the vulnerabilities of RL algorithms in different scenarios.

(3) We conduct a series of experiments for various environments and state-of-the-art deep policy-based RL algorithms, which demonstrates RL agents' vulnerabilities to even weaker attackers with limited knowledge and attack budget.

4.2 Related Work

The main focus of this chapter is on poisoning RL, an emerging area in the past few years.

Targeted Poisoning Attacks for RL. Most RL poisoning researches work on targeted poisoning, also called policy teaching, where the attacker leads the agent to learn a pre-defined target policy. Policy teaching can be achieved by manipulating the rewards [88, 89] or dynamics [35] of the MDP. However, they require the attackers to not only have prior knowledge of the environments (e.g., the dynamics of the MDP), but also have the ability to *alter the environment* (e.g. change the transition probabilities), which are often unrealistic or difficult in practice.

Poisoning RL with Omniscient Attackers. Most guaranteed poisoning RL literature [35, 86] assume *omniscient attackers*, who not only know the learner's model, but also know the underlying MDP. However, as motivated in the introduction, the underlying MDP is usually either unknown or too complex in practice. Some works poison RL learners by changing the reward signals sent from the environment to the agent. For example, Ma et al. [86] introduce a policy teaching framework for batch-learning model-based agents; Huang et al. [32] propose a reward-poisoning attack model, and provide convergence analysis for Q-learning; Zhang et al. [34] present an adaptive reward poisoning method for Q-learning (while it also extends to DQN) and analyze the safety thresholds of RL; these papers all assume the attacker knows not

only the models of the agent, but also the parameters of the underlying MDP, which could be possible in a tabular MDP, but hard to realize in large environments and modern deep RL systems.

On the contrary, *we consider non-omniscient attackers* who do not know the underlying MDP or environment in this chapter. The non-omniscient attackers can be further divided into two categories: *white-box* attackers, who know the learner's model/parameters, and *black-box* attackers, who do not know the learner's model/parameters. They both tap the interactions between the learner and the environment.

Black-box Poisoning for Value-based Learners. Although there are many successful black-box evasion approaches [26, 90], black-box poisoning in RL is rare. There is a *black-box* attacking method for a value-based learner (DQN) proposed by [31], which does not require the attacker to know the learner's model or the underlying MDP. In this work, the attacker induces the DQN agent to output the target action by perturbing the state with Fast Gradient Sign Method (FGSM) [20] in every step. However, the data-correlation problem of RL (Challenge II) is not considered, and FGSM attack does not work for policy-based methods due to their high stochasticity, as we show in experiments.

In this chapter, we propose a new poisoning algorithm for *policy-based deep RL* agents, which can achieve *both non-targeted and targeted* attacks. We do not require any prior knowledge of the environment. And our algorithm works not only when the attacker knows the learner's model (white-box), but also when the leaner's model is hidden (black-box).

4.3 Background and Problem Setup

4.3.1 Notations and Preliminaries

In RL, an agent interacts with the environment by taking actions, observing states and receiving rewards. The environment is modeled by a Markov Decision Process (MDP), which is denoted by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition kernel, R is the reward function, $\gamma \in (0, 1)$ is the discount factor, and μ is the initial state distribution. A *trajectory* $\tau \sim \pi$ generated by *policy* π is a sequence $s_1, a_1, r_1, s_2, a_2, \cdots$, where $s_1 \sim \mu$, $a_t \sim \pi(a|s_t)$, $s_{t+1} \sim P(s|s_t, a_t)$ and $r_t = R(s_t, a_t)$. The goal of an RL agent is to find an optimal policy π^* that maximizes the *expected total rewards* η, which is defined as $\eta(\pi) = \mathbb{E}_{\tau\sim\pi}[r(\tau)] = \mathbb{E}_{s_1,a_1,\cdots\sim\mu,\pi,P,R}[\sum_{t=1}^{\infty} \gamma^{t-1} r_t]$.

We use an overhead check sign $\check{}$ on a variable to denote that the variable is poisoned. For example, if the attacker perturbs a reward r_t, then the poisoned reward is denoted as $\check{r}_t$. If a policy π is updated with poisoned observation, then it is denoted as $\check{\pi}$.

4.3.2 The Procedure of Online Learning and Poisoning

Procedure of Online Learning. We consider a classical online policy-based RL setting, where the learner iteratively updates its *policy* π parametrized by θ, through K iterations with the environment. For notation simplicity, we omit θ and use π_k to denote π_{θ_k}, the learner's policy at iteration k. The online learning process is described as below.

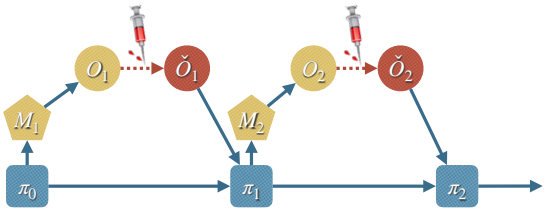


Figure 4.2: Online poisoning-learning process.

At iteration $k = 1, \cdots, K$,

(1) The agent uses the current policy π_k to roll out **observation** $\mathcal{O}_k = (\mathcal{O}_k^{\boldsymbol{s}}, \mathcal{O}_k^{\boldsymbol{a}}, \mathcal{O}_k^{\boldsymbol{r}})$ from environment $\mathcal{M}_k$, where $\mathcal{O}_k^{\boldsymbol{s}} = [s_1, s_2, \cdots], \mathcal{O}_k^{\boldsymbol{a}} = [a_1, a_2, \cdots], \mathcal{O}_k^{\boldsymbol{r}} = [r_1, r_2, \cdots]$ are respectively the sequence of states, actions and rewards generated at iteration k.

(2) The agent updates its policy parameters θ with its algorithm f. Most policy-based algorithms perform *on-policy* updating, i.e., update policy only by the current observation $\mathcal{O}_k$. The on-policy update can be then formalized as $\pi_{k+1} = f(\pi_k, \mathcal{O}_k) \approx \text{argmax}_\pi J(\pi, \pi_k, \mathcal{O}_k)$, where J is an objective function defined by algorithm f, e.g., the expected total reward $\eta(\pi)$.

Procedure of Online Poisoning. A poisoning attacker influences the learner in the training process by perturbing the training data. In SL, the training data consists of features and labels, and the attacker poisons the training data before the learning starts. However, the training data in RL is the trajectories a learner rolls out from the environment, i.e., observation $\mathcal{O} = (\mathcal{O}^{\boldsymbol{s}}, \mathcal{O}^{\boldsymbol{a}}, \mathcal{O}^{\boldsymbol{r}})$. At iteration k, the attacker eavesdrops on the interaction between the learner and the environment, obtains the observation $\mathcal{O}_k$, and may poison it into $\check{\mathcal{O}}_k$, then send $\check{\mathcal{O}}_k$ to the learner before policy updating. [1] Procedure 7 in Section 4.8.1 illustrates how the online game goes between the learner and the attacker. Figure 4.2 visualizes this online learning-poisoning procedure, where we can see that learning and poisoning are convoluted and inter-dependent.

[1] In this chapter, we assume the attacker poisons the observation(trajectories), which is the most universal setting in practice. Section 4.8.1 extends our problem formulation to a more general case where the attacker can change the underlying MDP.

4.4 Proposed Approach

4.4.1 A Unified Formulation for Poisoning Online Policy-based RL.

4.4.1.1 Attacker's Poison Aim

As defined in Section 4.3.1, the observation is a collection of trajectories, consisting of the observed states $\mathcal{O}^s$, the executed actions $\mathcal{O}^a$ or the received rewards $\mathcal{O}^r$. When poisoning the observation, the attacker may only focus on the states, or on the actions, or on the rewards. We call the quantity being altered as the **poison aim** of the attacker, denoted by $\mathcal{D} \in \{\mathcal{O}^s, \mathcal{O}^a, \mathcal{O}^r\}$. For example, $\mathcal{D} = \mathcal{O}^s$ means the attacker chooses to attack the states.

Distinguishing different poison aims is important, since in real-world applications different poison aims correspond to different behaviors of the attacker. For example, in an RL-based recommender system, the RL agent recommends an item (i.e., an action) for a user (i.e., a state), and the user may or may not choose to click on the recommended item (i.e., a reward). An adversary might manipulate the reward (poisoning $\mathcal{D} = \mathcal{O}^r$), e.g., blocking the user's click from the RL agent or creating a fake click. An adversary might also alter the state (poisoning $\mathcal{D} = \mathcal{O}^s$), e.g., raising a teenager user's age which could result in inappropriate recommendations. An adversary might also change the action (poisoning $\mathcal{D} = \mathcal{O}^a$), e.g., inserting a fake recommendation into the agent's list of recommendations. Under different scenarios, the feasibility of poisoning different aims may vary.

Most existing works on poisoning RL only solve one type of poison aim. Zhang et al. [34] and Huang et al. [32] propose to poison rewards, and Behzadan et al. [31] assume the attacker poison the states. However, in our paper, we provide a general solution for any of the poison aims

to satisfy the needs in different scenarios. Our proposed method also supports a "hybrid" poison aim, where the attacker could switch aims at different iterations, as discussed in Section 4.6.

4.4.1.2 A Poisoning Framework for RL

We focus on proposing a poisoning mechanism for the above challenging online learning scenario. We first formalize the poisoning attacking at iteration k as a *sequential bilevel optimization* problem in Problem (Q), and explain the details of the problem in the remaining of this section.

$$\underset{\check{\mathcal{D}}_k,\cdots,\check{\mathcal{D}}_K}{\operatorname{argmin}} \quad \sum_{j=k}^{K} \lambda_j L_A(\check{\pi}_{j+1}) \qquad \text{((a) attacker's weighted loss)} \qquad \text{(Q)}$$

$$s.t. \qquad \check{\pi}_{j+1} = \operatorname{argmax}_\pi J(\pi, \tilde{\pi}_j, \check{\mathcal{O}}_j | \check{\mathcal{D}}_j), \forall k \le j \le K \qquad \text{((b) imitate the learner)}$$

$$\sum_{j=1}^{K} \mathbf{1}\{\check{\mathcal{D}}_j \neq \mathcal{D}_j\} \le C \qquad \text{((c) limited-budget)}$$

$$U(\mathcal{D}_j, \check{\mathcal{D}}_j) \le \epsilon, \forall 1 \le j \le K \qquad \text{((d) limited-power)}$$

(a) Attacker's Weighted Loss. $L_A(\check{\pi})$ measures the attacker's loss w.r.t. a poisoned policy π. As the definition of poisoning implies, the attacker influences or misleads the learner's policy. $\lambda_{k:K}$ are the weights of future attacker losses, controlling how much the attacker value the poisoning results in different iterations. The goal of the attacker is either (1) **non-targeted poisoning**, which minimizes the expected total rewards of the learner, i.e., $L_A = \eta(\check{\pi})$, or (2) **targeted poisoning**, which induces the learner to learn a pre-defined target policy, i.e., $L_A = \text{distance}(\check{\pi}, \pi^\dagger)$, where $\text{distance}(\check{\pi}, \pi^\dagger)$ can be any distance measure between a learned policy $\check{\pi}$ and a target policy $\pi^\dagger$. Note that the targeted poisoning objective can usually be directly computed with a prior target policy, while non-targeted poisoning has a "reward-minimizing" objective, which is the reverse of the learner's objective. Without any prior knowledge of the environment, non-targeted

poisoning is usually more difficult, as the attacker needs to first learn "what is the worst way" (which is as difficult as a learning problem by a RL agent) and then lead the learner to that way (which is as difficult as a targeted poisoning problem, assuming leading an agent to different policies is roughly equally challenging). However, most existing poisoning work focus on targeted poisoning, which requires the attacker to know a pre-defined target policy. Thus in this chapter, we make more efforts to solve the reward-minimizing poisoning problem, which may deprave the policy without any prior knowledge.

(b) Imitate the Policy-Based Learner. To confidently mislead a learner, the attacker needs to predict how the learner will behave under the poison, which can be achieved by *imitating the learner using the learner's observation*. More specifically, at the j-th iteration, the attacker estimates the learner's policy to be $\tilde{\pi}_j$, called **imitating policy**. Then, the attacker predicts the next-policy $\check{\pi}_{j+1}$ under poisoned observation, based on the learner's update rule $\text{argmax}_\pi J(\pi, \tilde{\pi}_j, \check{\mathcal{O}}_j|\check{\mathcal{D}}_j)$, where $\mathcal{D} \in \{\mathcal{O}^{\boldsymbol{s}}, \mathcal{O}^{\boldsymbol{a}}, \mathcal{O}^{\boldsymbol{r}}\}$ stands for the poison aim of the poisoning, $\check{\mathcal{O}}|\check{\mathcal{D}}$ denotes that $\mathcal{O}$ is poisoned into $\check{\mathcal{O}}$ given that poison aim $\mathcal{D}$ is poisoned into $\check{\mathcal{D}}$. However, the imitating policy $\tilde{\pi}$ may or may not be the same as the actual learner's policy, depending on the *attacker's knowledge*. As introduced in Section 4.2, we deal with both white-box and black-box attackers, and both of them do not know the environment $\mathcal{M}$. A **white-box** attacker knows the current and past observations $\mathcal{O}_{1:k}$, the learner's algorithm f and policy π, so it can directly copy the policy $\tilde{\pi}_j = \pi_j, \forall j$. A **black-box** attacker knows the current and past observations $\mathcal{O}_{1:k}$, but does not know the learner's policy π. In this case, the attacker has to estimate π at every iteration. Section 4.5.3 states how to guess π.

(c,d) Limited-budget and Limited-power. In practice, the ability of an attacker is usually restricted by some constraints. For the online poisoning problem, we consider attacker's con-

straints in two forms: (1) (**attack budget** C) the total number of iterations that the attacker could poison does not exceed C; (2) (**attack power** ϵ) in one iteration, the total change[2] $U(\mathcal{D}_k, \check{\mathcal{D}}_k)$ between $\mathcal{D}_k$ and $\check{\mathcal{D}}_k$ can not be larger than ϵ. Attack power controls the amount of perturbation, as commonly used in the adversarial learning literature. Attack budget considers the frequency of attack, which is similar to the constraint studied by [85].

Problem (Q) is a generic formulation, covering a variety of poisoning models, and specifies the best poison an attacker can execute. However, directly solving Problem (Q) is prohibitive, as (1) the future observations $\mathcal{O}_{k+1:K}$ are unknown when poisoning the k-th iteration, as the attacker has no knowledge of the underlying MDP. (2) the limited-budget constraint is analogous to ℓ_0-norm regularization, which is generally NP-hard [91]; and (3) minimizing attacker's loss while maximizing learner's gain is non-convex minimax optimization, which is a complex problem [92].

In spite of the above difficulties, we introduce a practical method to approximately and effectively solve Problem (Q) in Section 4.5.

4.5 VA2C-P: Poison Policy Gradient Learners

In this section, we propose a practical and efficient poisoning algorithm called Vulnerability-Aware Adversarial Critic Poison (VA2C-P) for policy gradient learners. Without loss of generality, we assume the loss weights $\lambda_j = 1$ for all $j = 1, \cdots, K$.

Main Idea. As discussed in Section 4.4.1, Problem (Q) is difficult mainly because of the unknown future observations and the limited budget constraint. In other words, it is hard to ex-

[2]There are many choices of $U(\cdot, \cdot)$. For example, the total effort w.r.t. $\mathcal{O}^s$-poisoning can be the average ℓ_p-distance between any poisoned and unpoisoned state in $\mathcal{O}^s$ and $\check{\mathcal{O}}^s$.

actly determine (1) what kind of attack benefits the future the most, and (2) which iterations are worth attacking the most. Thus, we propose to break Problem (Q) into two decisions: when to attack, and how to attack. The "when to attack" decision allocates the limited budget to iterations which are more likely to be influenced by the attacker, and the "how to attack" decision utilizes limited power to minimize the attacker's loss. We introduce two mechanisms of VA2C-P, vulnerability-awareness and adversarial critic, to make these two decisions respectively.

4.5.1 Decision 1: When to Attack – Vulnerability-Aware

To answer the question of when to attack, we identify the iterations under which the learner's policy gets more depraved by the same level of attack power. Inspired by the notion of stability in learning theory, which measures how a machine learning algorithm changes due to a small perturbation of the input data, we formally investigate the stability of an RL algorithm, which is the first attempt in the existing literature to the best of our knowledge.

Stability of RL Algorithms. We first focus on a single update process of an algorithm f. An update $\pi' = f(\pi, \mathcal{O})$ is stable if a limited-power poisoning attack does not cause any difference on the output policy π'. That is, the learning algorithm produces the same result regardless of the presence of the poison. More formally, we define the concept of *stability radius of one update* in Definition 33.

Definition 33 (Stability Radius of One Update)**.** *For the update of an RL algorithm $\pi' = f(\pi, \mathcal{O})$, with any poison aim $\mathcal{D}$, the δ-stability radius of the update is defined as the minimum poison*

power needed to cause δ change in policy (called δ-policy-discrepancy)

$$\phi_{\delta,\mathcal{D}}(f,\pi,\mathcal{O}) = \inf_{\epsilon}\{\exists \check{\mathcal{D}} \textit{ s.t. } U(\mathcal{D},\check{\mathcal{D}}) \leq \epsilon \textit{ and } d^{\max}[\pi'||\check{\pi}'] > \delta, \textit{ where } \check{\pi}' = f(\pi,\check{\mathcal{O}}|\check{\mathcal{D}})\}, \tag{4.1}$$

Policy discrepancy $d^{\max}[\pi_1||\pi_2] = \max_s d\big[\pi_1(\cdot|s)||\pi_2(\cdot|s)\big]$, *where* $d[\cdot||\cdot]$ *could be any measure of distribution distance.*

Remarks. (1) The one-update stability radius is w.r.t. the algorithm f, the old policy π, the clean observation $\mathcal{O}$ and the poison aim $\mathcal{D}$. (2) Poison with power under $\phi_{\delta,\mathcal{D}}(f,\pi,\mathcal{O})$ will not cause the policy distributions to change more than δ. (3) As shown by Proposition 34 in Section 4.8.2.1, poison with power under $\phi_{\delta,\mathcal{D}}(f,\pi,\mathcal{O})$ will not make the policy value drop more than $O\big(\delta^2\gamma(1-\gamma)^{-2}\max_{s,a}|A_{\pi'}(s,a)|\big)$, where A is the advantage function, i.e., $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$.

Stability radius measures the minimal effort needed to make the *poisoned next-policy* $\check{\pi}'$ notably different from the *clean next-policy* π' which the learner will get if no poison is applied. Assuming $\max_{s,a}|A_{\pi'}(s,a)|$ does not drastically change for the learner's policy during training, then an attack that causes higher policy discrepancy between π' and $\check{\pi}$ could cause more drop of the policy value. Therefore, the idea of vulnerability-aware attack is to estimate the vulnerability of each update and attack the most vulnerable ones. More specifically, if the attacker finds an ϵ-powered attack results in a policy discrepancy larger than some threshold δ, then it can conclude $\phi_\delta(f,\pi,\mathcal{O}) \leq \epsilon$, and the current update is relatively vulnerable. In practice, we have the fixed budget C instead of δ, so we perform the vulnerability check in an adaptive way: attacking the C iterations where ϵ-powered attacks can trigger the highest policy discrepancies. Section 4.5.3 introduces an algorithm to realize this idea.

4.5.2 Decision 2: How to Attack – Adversarial Critic

Since "when to attack" decision tackles the limited-budget constraint, now we turn our attention to minimizing the attacker's loss while satisfying the limited-power constraint. If the attacker has already decided to poison the k-th iteration due to its high vulnerability, then the decision of how to attack should be made before the learner uses the observation to update the policy. We relax the original Problem (Q) into Problem (P) as below.

$$\begin{aligned} &\text{argmin}_{\check{\mathcal{D}}_k} && L_A(\check{\pi}_{k+1}) && \text{(P)} \\ &s.t. && \check{\pi}_{k+1} = \text{argmax}_\pi J(\pi, \tilde{\pi}_k, \check{\mathcal{O}}_k | \check{\mathcal{D}}_k) \\ & && U(\mathcal{D}_k, \check{\mathcal{D}}_k) \leq \epsilon \end{aligned}$$

Compared with Problem (Q), Problem (P) does not consider future losses, which require the unavailable future observations. Instead, Problem (P) finds a greedy attack $\check{\mathcal{D}}_k$ to minimize the loss of the immediate next iteration. The solution to Problem (P) is always feasible to Problem (Q), although might not be optimal.

In practice, it is also challenging to estimate $L_A(\check{\pi})$. For targeted attacking, $L_A = \text{distance}(\check{\pi}, \pi^\dagger)$ is directly computable with a properly defined distance metric, but for non-targeted attacking, the loss $L_A = \eta(\check{\pi})$ can not be directly computed, since $\check{\pi}$ is not the behavior policy. Although one can use importance sampling to evaluate $\eta(\check{\pi})$ with the current trajectories generated by the learner's policy, i.e., $\mathbb{E}_{\tau \sim \pi_{k-1}}[\frac{\pi_k(\tau)}{\pi_{k-1}(\tau)} r(\tau)]$, it may suffer from a high variance [93] when there are few trajectories. To solve this challenge, we introduce another mechanism, adversarial critic.

Adversarial Critic. Under poisoning, the value network (if any) held by the learner usually fails to fit the correct value of its policy, since it does not observe the real trajectories generated by its policy. However, the attacker observes the real trajectories before poisoning,

and is able to learn the real values to make the attacking stronger. Inspired by the Actor-Critic method, we propose to let the attacker learn a value function (network) $\tilde{V}_\omega$ with observations of the learner, i.e., the attacker learns a critic of the learner's current policy π. Then the attacker can use $\tilde{V}_\omega$ to design poisoned observations, directing the learner to a decreasing-value direction, which is called *Adversarial Critic*. Then, using importance sampling, the attacker's loss becomes $\mathbb{E}_{s,a\sim\pi_k}[\frac{\check{\pi}(a|s)}{\pi_k(a|s)}\big(G(s_t,a_t)-\tilde{V}_\omega(s_t)\big)]$, where G is the discounted future reward $\sum_{i=t}^{T}\gamma^{i-t}r_t$.

4.5.3 Poisoning Algorithm VA2C-P

Algorithm 6: Vulnerability-Aware Adversarial Critic Poison

Input: total iterations of learning K; poisoning power ϵ; poisoning budget C; Initialize a list of policy discrepancies $\Psi=\emptyset$, the number of poisoned iterations $c=0$
Initialize an adversarial critic network $\tilde{V}_\omega$ and an imitating policy network $\tilde{\pi}$
for $k=1,\cdots,K$ **do**
  if $c>C$ **then**
    Break
  Obtain the observation (trajectories) $\mathcal{O}_k$ obtained by the learner
  Update the adversarial critic $\tilde{V}_\omega$ with observation $\mathcal{O}_k$
  if *White-box* **then**
    Copy the learner's policy parameters to the imitating policy of attacker: $\tilde{\pi}\leftarrow\pi_k$
  Compute the clean next-policy $\pi'\leftarrow f(\tilde{\pi},\mathcal{O}_k)$
  Solve Problem (P) with power ϵ and critic $\tilde{V}_\omega$, poison $\mathcal{O}_k$ to $\check{\mathcal{O}}_k$
  Compute the poisoned next-policy $\check{\pi}'\leftarrow f(\tilde{\pi},\check{\mathcal{O}}_k)$
  Estimate the policy discrepancy $\widehat{\psi}_k$ between π' and $\check{\pi}'$
  Add $\widehat{\psi}_k$ to the list of policy discrepancies Ψ
  if $\widehat{\psi}_k\geq\lfloor\frac{C-c}{K-k}\rfloor$*-th largest element in* Ψ **then**
    Send the learner the poisoned observation $\check{\mathcal{O}}_k$
    if *Black-box* **then**
      Update the imitating policy as the clean next-policy: $\tilde{\pi}\leftarrow\pi'$
  else
    Send the clean observation $\mathcal{O}_k$
    if *Black-box* **then**
      Update the imitating policy as the poisoned next-policy: $\tilde{\pi}\leftarrow\check{\pi}'$

Algorithm 6 illustrates how an attacker can poison an online RL learner with our proposed VA2C-P, which is corresponding to Line 4 in Procedure 7 shown in Section 4.8.1 . More implementation details are in Algorithm 8 in Section 4.8.5.

In an iteration, an attacker first finds a good perturbation for observation $\mathcal{O}$ with the current attack power ϵ, then estimates how much the output policy will change by computing the policy discrepancy between the clean next-policy and the poisoned next-policy. Then the attacker poisons if the policy discrepancy ranks high in the historical policy discrepancies, considering the remaining iterations and budget (Line 15). In Line 11, we use projected gradient descent to solve Problem (P). Computation details are illustrated in Section 4.8.5.

Algorithm 6 covers both white-box and black-box settings, both of which maintain an imitating policy $\tilde{\pi}$ to keep track of the learner's potential status. As mentioned in Section 4.4.1, a white-box attacker knows the learner's current policy, thus he can directly copy the learner's parameters to his imitating policy (Line 9), then predict the next-policy the learner would get under different observations. In contrast, a black-box attacker does not know the learner's policy, but he can update its imitating policy using the same observation as the learner uses (Line 18, 22). As claimed and verified by many black-box attacking methods [31], adversarial attacks are usually transferable, i.e., if the attack works on the imitating learner, then the attack is also likely to work for the real learner. Note that as assumed by [31], the black-box attacker knows what RL algorithm the learner is using (e.g., PPO, A2C, etc), so that the black-box attacker computes its estimation for the next-policy in Line 10 and Line 12.

4.6 Empirical Results

In this section, we evaluate the performance of VA2C-P by poisoning multiple algorithms on various environments. We demonstrate that VA2C-P can effectively reduce the total reward of a training agent, or force the agent to choose a specific policy with limited power and budget. Moreover, VA2C-P works for heterogeneous poison aims, and works in both white-box and black-box settings.

Experiment Setup. We choose 4 policy-gradient learning algorithms, including Vanilla Policy Gradient [94], A2C [41], ACKTR [42] and PPO [16]. And we choose 5 Gym [95] environments with increasing difficulty levels: CartPole, LunarLander, Hopper, Walker and HalfCheetah. All results are averaged over 10 random seeds. The total effort U is calculated by the normalized ℓ_2-distance. See Section 4.8.6.1 for the expression of U, as well as more hyper-parameter settings.

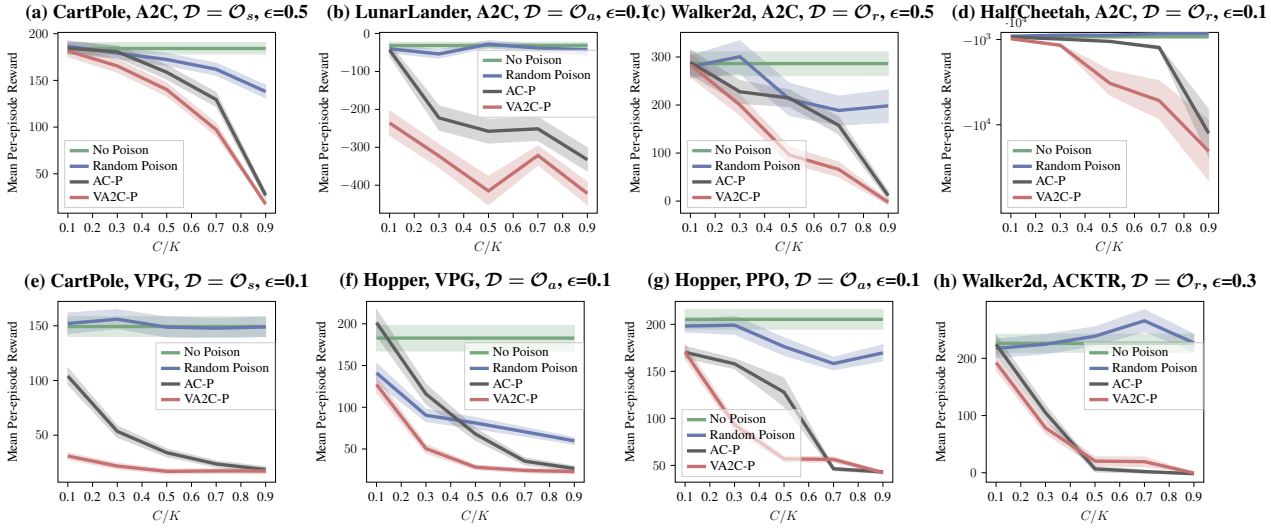


Figure 4.3: Comparison of mean per-episode reward gained by VPG, PPO, A2C, ACKTR on various environments, under no poisoning, random poisoning, AC-P and VA2C-P.

Reward-minimizing Poisoning. *Baselines.* To the best of our knowledge, there is *no*

existing poisoning algorithm against deep policy-gradient algorithms. Although some evasion methods [26] also work for policy-gradient algorithms, evasion is substantially different from poisoning since it does not influence the policy. Therefore, to show the effectiveness of VA2C-P, we compare it with 3 baselines: (1) the normal learning with no poison; (2) a random attacker which randomly chooses C iterations, and perturbs the reward to an arbitrary direction by ϵ; and (3) a simplified version of our algorithm, called Adversarial Critic Poison (AC-P), which decides "how to attack" in the same way as VA2C-P does, but chooses "when to attack" randomly.

Performance. We first show the reward-minimizing performance of VA2C-P on all three types of poison aims, assuming the attacker knows the learner's model (white-box attack). Figure 4.3 shows the rewards of various learners under different kinds of poisoning methods, with different ratios of budget C to the total number of iterations K. Compared with random poisoning, our proposed *VA2C-P and the simplified version AC-P make the reward drop more significantly*, demonstrating the effectiveness of our "how to attack" decisions made by the Adversarial Critic. VA2C-P further outperforms AC-P in almost all cases, which implies that our "when to attack" decisions based on Vulnerability-Awareness work well in practice. An interesting observation is *random poisoning not only does not work well in many cases, but sometimes also facilitates the learner* (Figure 4.3h). This phenomenon is mainly due to the uncertainty of the environment, as pointed out by *Challenge I* and *Challenge II* in Section 4.1. Thus, a good poisoning strategy is important.

Black-box Poisoning. Figure 4.3 demonstrates the performance of VA2C-P when the attacker knows the learner's model. But when the learner's model is not available (black-box attack), VA2C-P could also work as shown in Figure 4.4a, where one can see that *black-box poisoning is still effective, although worse than white-box poisoning due to the lack of knowledge*.

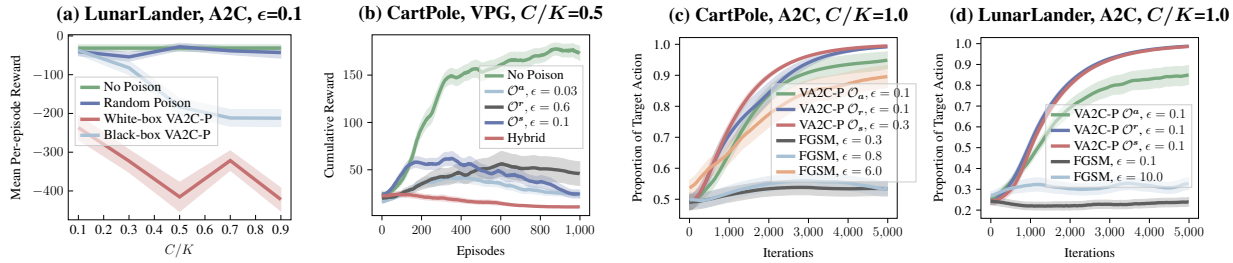


Figure 4.4: (a) VA2C-P also works under black-box setting; (b) hybrid-aim poisoning could be better than single-aim poisoning; (c)(d) VA2C-P successfully forces the agent to choose the target policy.

Hybrid-aim Poisoning. The experiments shown in Figure 4.3 assume that the attacker poisons one of the 3 poison aims, $\mathcal{O}^s$, $\mathcal{O}^a$ or $\mathcal{O}^r$. But if the attacker happens to have access to all of these poison aims, it can do better by performing a "hybrid" poisoning, i.e., at each iteration, evaluate the vulnerability of the algorithm w.r.t. each poison aim with their corresponding attacker power, then choose the one with the highest vulnerability. Figure 4.4b shows that *adaptive attacking using a hybrid poison aim could significantly outperform attacking using a fixed single poison aim.*

Targeted Poisoning. *Baselines.* Although targeted RL poisoning is studied by many works [32, 35], few of them work for deep RL. Despite that no existing work focuses on deep policy-based methods, we transfer the FGSM-based targeted poisoning method proposed for DQN by [31] to attack policy-based learners as our baseline. The attacking method is, for each new state, adding a perturbation to it such that the perturbed state is pushed across the decision boundary and towards the target action. See Section 4.8.6.2 for details.

Performance. Figure 4.4c and 4.4d respectively show the targeted-attack performance comparison on CartPole and LunarLander, where the target policies are both "always going to the right". The y-axes show the proportion of target actions among all actions taken by the learner.

VA2C-P successfully leads the learner to learn the target policy using all types of poison aims with relatively small attack power. In contrast, FGSM fails to let the learner select the target action in most cases, even with much larger attack power. The main reason why FGSM works for DQN [31] but not for policy-gradient methods is, policy-based agents generally learn stochastic policies, and even though FGSM could perturb the state such that the output probability for the target action is 0.51, the agent will still choose other actions with probability 0.49. Therefore, *FGSM cannot easily perform targeted poisoning against policy-based learners.*

More experiment results are in Section 4.8.6.3. Note that our proposed poisoning method can also be extended to off-policy learners, as discussed in Section 4.8.6.4.

4.7 Conclusion

Poisoning RL is a difficult problem due to the uncertainty of the environment. In this chapter, we build a generic poisoning framework for online RL with an in-depth comparison with SL, and propose VA2C-P, the first generic poisoning algorithm for deep policy-gradient online RL methods, which incorporates heterogeneous poisoning models and does not require any prior knowledge of the environment. By systematic experiments, we show the effectiveness of VA2C-P in multiple settings and for various state-of-the-art RL algorithms. We first formalize the online RL poisoning problem as a hard sequential bi-level optimization problem, then reduce it into two decisions: when to attack and how to attack. Two mechanisms, vulnerability-awareness and adversarial critic are proposed to solve these two decisions. algorithm that minimize the agent's reward with limited attacking budget. In addition, we introduce stability radius, a novel metric to measure the vulnerability of RL algorithms at training time. We also provide insights on the

differences and relations between SL and RL. In this chapter, we discuss the scenario where the attacker only poisons one type of target. But it could also be extended to a simultaneous poisoning on multiple target types. Ultimately, by studying the attacks in RL, we aim to understand the vulnerability of RL agents and finally design robust RL agents that can combat adversarial attacks. Although the effectiveness of VA2C-P is verified in a wide range of RL algorithms and environments, we acknowledge that poisoning RL in practice is still challenging, due to the relatively high computational burden and the uncertainty of the environments. These challenges are also exciting opportunities for future work.

4.8 Supplemental Materials: Proofs and Additional Details

4.8.1 A Generic Poisoning Framework for RL

In this section, we establish a generic framework of poisoning in online RL, systematically characterizing its challenges and difficulties from multiple perspectives - objective of poisoning, various poison aims, and attacker's knowledge. Our in-depth comparison with the SL allows a thorough understanding of the additional vulnerability of online RL systems compared with the well-understood SL systems. Our framework also provides a clear context to correctly position prior works in the literature as well as to compare our work with existing works. Compared to the poisoning framework described by [32], we provide a solution for unifying these poison aims in one attack model in this section.

We consider the online learning scenario, where the RL agent (the learner) does not know the dynamics or rewards of the underlying MDP $\mathcal{M}$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics P, rewards R and discount factor γ.

Settings and Notations In online RL, the learner interacts with the environment and collects observations. The learner's algorithm, denoted by f, iteratively searches for a policy π parametrized by θ, through K interactions with the environment. Before learning starts, the learner initializes a policy π_1. At each iteration k, the learner uses its previous policy π_{k-1} to roll out *observations* $\mathcal{O}_k$ from the MDP $\mathcal{M}$. $\mathcal{O}_k$ is a concatenation of multiple *trajectories*, denoted as $\mathcal{O}_k = (\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{r}_k)$, where $\boldsymbol{s}_k = [s_1, s_2, \cdots], \boldsymbol{a}_k = [a_1, a_2, \cdots], \boldsymbol{r}_k = [r_1, r_2, \cdots]]$ are respectively the sequence of states, actions, and rewards in iteration k. Then, with the observations $\mathcal{O}_k$, the learner updates its policy by attempting to solve $\text{argmax}_\pi J(\pi, \pi_{k-1}, \mathcal{O}_k)$, where J is the objective function. The generated policy by the learner's algorithm $\pi_k = f(\pi_{k-1}, \mathcal{O}_k)$[3] does not necessarily achieve the maximization of the objective function.

Algorithm 7: Flow of Online RL Poisoning()

1 Learner initializes its initial policy π_{θ_0}
2 **for** $k = 1, \cdots, K$ **do**
3 Learner rollouts observation (trajectories) $\mathcal{O}_k$ in environment $\mathcal{M}$ with current policy π_k
4 Attacker may poison the observation $\mathcal{O}_k$ to $\check{\mathcal{O}}_k$
5 Learner updates its policy: $\pi_{k+1} = f(\pi_k, \check{\mathcal{O}}_k) \approx \text{argmax}_\pi J(\pi, \pi_k, \check{\mathcal{O}}_k)$

In this chapter, an overhead check sign $\check{}$ on a variable always denotes that the variable is poisoned. For example, if the attacker changes a reward r_t, then the poisoned reward is denoted as $\check{r}_t$.

Poison Objective. We use L_A to denote loss function of the poisoning attack, which the attacker attempts to minimize. The form of L_A is determined by its goal, which falls into one of the two categories, *non-targeted* and *targeted* poisoning.

In *non-targeted poisoning*, the attack poisons a policy π to $\check{\pi}$ to minimize the learner's

[3]For algorithms with experience replay, the update can be extended as $\pi_{k+1} = f(\pi_k, \mathcal{O}_{1:k})$.

expected rewards. Therefore the poison objective L_A is to minimize the learner's value $\eta(\check{\pi})$.

In *targeted poisoning*, the attack "teaches" the agent to learn a pre-defined target policy $\pi^\dagger$. Therefore the poison objective L_A is defined as distance[4]$(\check{\pi}, \pi^\dagger)$.

Most existing poison RL researches focus on targeted poisoning [35, 86], and non-targeted poisoning, although discussed by [32], remains relatively untouched.

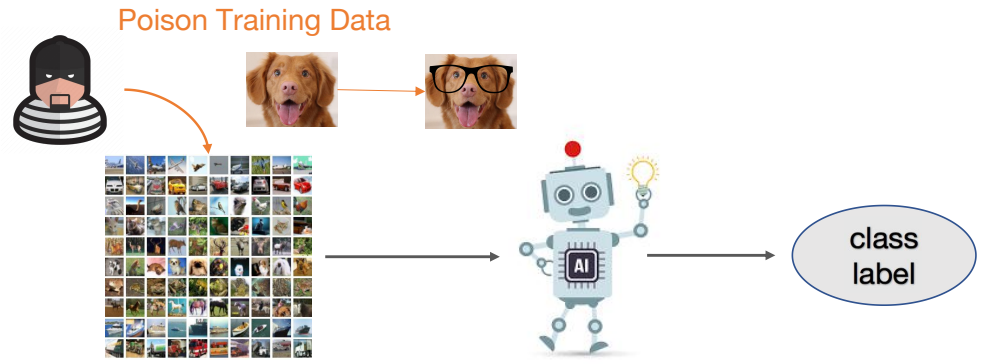
poison aims. To influence the behaviors of the learner, an attacker could inject poison at multiple locations of the learner's learning process as detailed in Figure 4.5. Part of the reason why poisoning in RL is more challenging than in SL is that it involves more poison aims of poisoning, some of which adapt with the environment, increasing the uncertainty.

Poison Aim I – Poison Observation $(\mathcal{O}_{\boldsymbol{r}}, \mathcal{O}_{\boldsymbol{s}})$. The attacker could manipulate the observation of the learner, i.e., change $\mathcal{O}$ into $\check{\mathcal{O}}$. This may happen when the attacker is able to intercept the communication between the learner and the environment, similar to the man-in-the-middle attack in cryptography. The attacker could target the rewards, called $\mathcal{O}_{\boldsymbol{r}}$*-poisoning*, studied by [32]; or the states, called $\mathcal{O}_{\boldsymbol{s}}$*-poisoning*, investigated by [31].

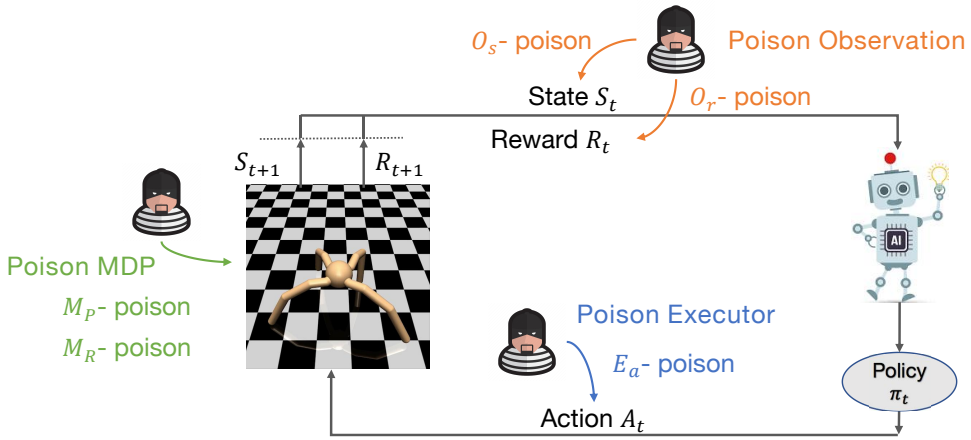
Poison Aim II – Poison MDP $(\mathcal{M}_R, \mathcal{M}_P)$. An attack could directly change the MDP (environment) that the learner is interacting with, i.e., change $\mathcal{M}$ into $\check{\mathcal{M}}$. For example, a seller could influence the behaviors of customers by changing the prices of products. The poison of MDP could be injected at the reward model R or the transition dynamics P, respectively denoted as $\mathcal{M}_R$*-poisoning* (studied by [86]) and $\mathcal{M}_P$*-poisoning* (studied by [35]) . The analogy of poison MDP in SL is to manipulate the underlying data distribution of the training data.

Poison Aim III – Poison Executor $E_{\boldsymbol{a}}$. The executor of the learner could be poisoned. For

[4]There are many ways to define the distance between two policies, for instance KL-divergence for stochastic policies [76], and average mismatch for deterministic policies [35].



(a) supervised learning



(b) reinforcement learning

Figure 4.5: Different poison aims of poisoning in supervised learning and reinforcement learning.

example, an attacker applies a force to the agent, so that the intended action "north" becomes "northeast". Pinto et al. [36] train a robust RL agent against the executor poisoner. Denote this type of poisoning as $E_{\boldsymbol{a}}$-*poisoning*. Note that $E_{\boldsymbol{a}}$-poisoning is equivalent to directly changing $\boldsymbol{a}$ stored in the observation $\mathcal{O} = (\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{d})$.

Attacker's Knowledge At the k-th iteration, what an attacker can do depends on its current knowledge set, denoted by $\mathcal{K}_k$. $\mathcal{K}_k$ could contain the underlying MDP $\mathcal{M}$, the learner's algorithm f, the learner's previous policy models $\theta_{1:k-1}$ as well as the previous and current observations $\mathcal{O}_{1:k}$.

An *omniscient attacker* knows everything , i.e, $\mathcal{K}_k^{(O)} = \{\mathcal{M}, f, \theta_{1:k-1}, \mathcal{O}_{1:k}\}$. Most guaranteed policy teaching literature [35, 86] assume omniscient attacker. However as motivated in the introduction, it is often unrealistic to exactly know the underlying environment. We discuss two more realistic setting where the attacker only has limited knowledge as follows.

A *monitoring attacker* has some information but does not assume knowledge of the underlying MDP $\mathcal{M}$, i.e., $\mathcal{K}_k^{(M)} = \{f, \theta_{1:k-1}, \mathcal{O}_{1:k}\}$. This is especially relevant in applications where learner's information is not secure (or even open), or an attacker hacks to steal information from the learner. Monitoring attacker is similar to the white-box attacker in supervised learning.

A *tapping attacker* has very limited knowledge and knows the observations only, i.e., $\mathcal{K}_k^{(T)} = \{\mathcal{O}_{1:k}\}$. This is widely applicable since the tapping the communication between the learner and the environment is easy. Tapping attacker is analogous to the black-box attacker in supervised learning. [31] consider a tapping attacker, which observes and manipulates the states but does not know the learner's parameters.

4.8.2 Theoretical Analysis

4.8.2.1 Measuring Training-Time Vulnerability of RL Algorithms via Stability Radius

Definition 33 in Section 4.5 measures the stability of one update based on policy discrepancy. But it is not obvious whether the rewards change drastically due to the attacks. Proposition 34 provides a guarantee on the performance of the poisoned policy $\check{\pi}'$, compared with the un-poisoned new policy π'.

Proposition 34. *For an update of stochastic policy $\pi' = f(\pi, \mathcal{O})$, if its δ-stability radius is ε with the total variance measure, then any poisoning effort smaller than ε on $\mathcal{D}$ will cause the expected total reward drop ($\eta(\pi') - \eta(\check{\pi}')$) by no more than*

$$\frac{4\delta^2\gamma \max_{s,a} |A_{\pi'}(s,a)|}{(1-\gamma)^2} + 2\delta \sum_{s\in\mathcal{S}} g_{\pi'}(s) \max_a A_{\pi'}|(s,a)| \tag{4.2}$$

where γ is the discount factor, $g_\pi(s) := \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi)$ is the discounted visitation frequency, and A is the advantage function, i.e., $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$.

Proposition 34 shows that if the attack power is within the stability radius, the reward of the poisoned policy will not be influenced too much. which also explains the motivation behind our proposed vulnerability-aware attack. The proof is in Section 4.8.4.1.

With Definition 33, the one-update stability measure, we are able to formally define the stability radius of an RL algorithm w.r.t. an MDP.

Definition 35 (Stability Radius w.r.t an MDP)**.** *The δ-stability radius of an algorithm f in an MDP $\mathcal{M}$ is defined as the minimal stability radius of all observations drawn from the MDP, and*

all possible policies in policy space Π. *If* f *is on-policy, then* $\phi(f, \mathcal{M}) = \min_{\pi \in \Pi, \mathcal{O} \sim \pi} \phi(f, \pi, \mathcal{O})$*; if* f *is off-policy, then* $\phi(f, \mathcal{M}) = \min_{\pi \in \Pi, \mathcal{O} \sim \overline{\pi}} \phi(f, \pi, \mathcal{O})$*, where* $\overline{\pi}$ *is the behavior policy;*

4.8.3 Measuring Test-Time Vulnerability of RL Algorithms via Robustness Radius

Different with poisoning, test-time evasion (adversarial examples) misleads the agent by manipulating the states only, since the agent no longer learns from interactions and feedbacks. Note that although Gleave et al. [25] propose an attack called "adversarial policy", the perturbation does not happen in the policy, but still happens in the input states (observations) of the agent.

To study how robust a trained policy is, we define the robustness radius with regard to both a single state and the whole environment.

Definition 36 (Robustness Radius of Policy w.r.t. a State)**.** *The robustness radius of a deterministic policy* π *on a state* s *is defined as the minimal perturbation of* s *which changes the output action, i.e.,*

$$(\pi, s) = \inf_{\varepsilon} \{\exists \check{s} \in \mathcal{S} \cap \mathcal{B}_{\varepsilon}(s) \quad s.t. \quad \pi(s) \neq \pi(\check{s})\} \tag{4.3}$$

Similarly, for any $0 < \delta < 1$*, the* δ*-robustness radius of a stochastic policy* π *on a state* s *is defined as the minimal perturbation of* s *which makes the output action distribution disagrees with the original* $\pi(s)$ *with probability more than* δ*, i.e.,*

$$(\pi, s)_{\delta} = \inf_{\varepsilon} \{\exists \check{s} \in \mathcal{S} \cap \mathcal{B}_{\varepsilon}(s) \text{ s.t. } d\big(\pi(\cdot|s)||\pi(\cdot|\check{s})\big) > \delta\} \tag{4.4}$$

where $d(\cdot|\cdot)$ *could be any distance measure between two distributions.*

Remark. If we regard the policy as a classifier which "classifies" a state to an action, then the robustness radius of policy defined above is analogous to the robustness radius of classifiers defined by [96], with an extension to stochastic predictions.

Definition 37 (Robustness Radius w.r.t an MDP)**.** *The (δ-)robustness radius of a policy π in an MDP $\mathcal{M}$ is defined as the maximal robustness radius of all states, i.e.,* $(\pi, \mathcal{M}) = \min_{s \in \mathcal{S}} (\pi, s)$

Remarks. (1) A deterministic policy is robust against any state perturbation smaller than $(\pi, \mathcal{M})$.

(2) For a stochastic policy, if its δ-robustness radius in an $\mathcal{M}$ is ε, then any state perturbation within ε will cause the expected total reward drop by no more than

$$(\frac{2\delta\gamma}{(1-\gamma)^2} + 2\delta) \max_{s,a} |R(s,a)|, \tag{4.5}$$

which is proven by Theorem 5 in [19].

4.8.4 Vulnerability Comparison: Difference Between SL and RL

To shed some light on understanding adversarial attacks in RL, we compare SL and RL in terms of their vulnerability to poisoning and adversarial examples.

At test time, a policy network receives states as input, and returns probabilities of choosing each actions as output; a value network receives states (or state-action pairs) as input, and returns the corresponding value as the output. Thus, test-time RL systems are very similar to SL systems, as one can view the policy networks as classification networks, and value networks as regression networks. However, the key difference between evasion in RL and evasion in SL is, data samples are not independent in RL. A single adversarial example in SL test dataset may cause at most one misclassification instance, whereas an adversarial example in RL may case a drastic change of

the gained rewards (e.g., by leading the agent to a "devastating" or "absorbing" state).

At training time, SL systems and RL systems are significantly different, as Figure 4.5 shows. Even when the supervised learner also learns from data streams in an online manner, the training data are independent with the learner's classifier. In contrast, the distribution of training data samples changes as the learner updates its policy. Poisoning attacks against an SL system could alter the decision boundary, so that the learner makes wrong decisions for certain data samples. For an RL system, poisoning attacks could (1) alter the decision boundary so that the learner chooses bad actions for certain states, and also (2) change the following observations and interactions due to a different selection of action.

In summary, an adversarial attacker may cause higher damages on RL systems than on SL systems, with the same power and budget. But it does not suggests attacking RL systems is easier than attacking SL systems. As every coin has two sides, the high uncertainty of the environment may help an attack reduce the learner's reward, but may also lead the learner to gain higher reward in the future (as shown in Section 4.6). Therefore, it is more challenging to successfully attack RL systems than SL systems with a specific goal.

4.8.4.1 Proof of Proposition 34

Proof. According to the definition of δ-stability radius, for any poisoning effort within ε, the poisoned policy satisfies $D_{TV}^{\max}[\pi' || \check{\pi}'] \leq \delta$ (assume total variance D_{TV} is the distance measure between policy distributions). We are interested in the difference of expected rewards $\eta(\pi') - \eta(\check{\pi}')$.

Define $L_{\pi'}(\check{\pi}') = \eta(\pi') + \sum_{s \in \mathcal{S}} g_{\pi'}(s) \sum_{a \in \mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a)$, where g is the discounted

state visitation frequencies, i.e.,

$$g_{\pi'}(s) := P\left(s_0 = s|\pi'\right) + \gamma P\left(s_1 = s|\pi'\right) + \gamma^2 P\left(s_2 = s|\pi'\right) + \cdots .$$

Since $D_{TV}^{\max}[\pi'||\check{\pi}'] \leq \delta$, follow Theorem 1 in paper [76], we can get

$$|\eta(\check{\pi}') - L_{\pi'}(\check{\pi}')| \leq \frac{4\delta^2\gamma \max_{s,a} |A_{\pi'}(s,a)|}{(1-\gamma)^2}. \tag{4.6}$$

So we have

$$|\eta(\check{\pi}') - \eta(\pi') - \sum_{s\in\mathcal{S}} g_{\pi'}(s) \sum_{a\in\mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a)| \leq \frac{4\delta^2\gamma \max_{s,a} |A_{\pi'}(s,a)|}{(1-\gamma)^2}, \tag{4.7}$$

which can be transformed to

$$\eta(\pi') - \eta(\check{\pi}') \leq \frac{4\delta^2\gamma \max_{s,a} |A_{\pi'}(s,a)|}{(1-\gamma)^2} - \sum_{s\in\mathcal{S}} g_{\pi'}(s) \sum_{a\in\mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a). \tag{4.8}$$

We upper bound the term $-\sum_{s\in\mathcal{S}} g_{\pi'}(s) \sum_{a\in\mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a)$ as below.

$$\begin{aligned}
&- \sum_{s\in\mathcal{S}} g_{\pi'}(s) \sum_{a\in\mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a) \\
=&\sum_{s\in\mathcal{S}} g_{\pi'}(s) \big(- \sum_{a\in\mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a) \big) \\
=&\sum_{s\in\mathcal{S}} g_{\pi'}(s) \big(- \sum_{a\in\mathcal{A}} \check{\pi}'(a|s) A_{\pi'}(s,a) + \sum_{a\in\mathcal{A}} \pi'(a|s) A_{\pi'}(s,a) - \sum_{a\in\mathcal{A}} \pi'(a|s) A_{\pi'}(s,a) \big) \\
=&\sum_{s\in\mathcal{S}} g_{\pi'}(s) \big(\sum_{a\in\mathcal{A}} A_{\pi'}(s,a) (\pi'(a|s) - \check{\pi}'(a|s)) \big) - \sum_{a\in\mathcal{A}} \pi'(a|s) A_{\pi'}(s,a) \big) \\
\leq&\sum_{s\in\mathcal{S}} g_{\pi'}(s) \big(2\delta \max_{a\in\mathcal{A}} |A_{\pi'}(s,a)| - \sum_{a\in\mathcal{A}} \pi'(a|s) A_{\pi'}(s,a) \big) \\
=&2\delta \sum_{s\in\mathcal{S}} g_{\pi'}(s) \max_{a\in\mathcal{A}} |A_{\pi'}(s,a)|
\end{aligned} \tag{4.9}$$

Combining the above results, we obtain

$$\eta(\pi') - \eta(\check{\pi}') \leq \frac{4\delta^2\gamma \max_{s,a} |A_{\pi'}(s,a)|}{(1-\gamma)^2} + 2\delta \sum_{s\in\mathcal{S}} g_{\pi'}(s) \max_{a} |A_{\pi'}(s,a)| \tag{4.10}$$

□

4.8.5 Algorithm Details

Assume the learner's policy π is parametrized by θ.

How to solve the projected gradient descent. To solve (P), assume $\frac{\partial \theta_k}{\partial \check{\boldsymbol{r}}}$ exists, one can use projected gradient descent to update $\boldsymbol{r}$ by using the chain rule:

$$\frac{\partial \eta_{\pi_{\theta_{k-1}}}(\pi_{\theta_k})}{\partial \check{\boldsymbol{r}}} = \frac{\partial \eta_{\pi_{\theta_{k-1}}}(\pi_{\theta_k})}{\partial \theta_k} \frac{\partial \theta_k}{\partial \check{\boldsymbol{r}}}.$$

For Vanilla Policy Gradient (VPG) whose update rule is, $\theta_k = \theta_{k-1} + \alpha \nabla_{\theta_{k-1}} \hat{\eta}(\pi_{\theta_{k-1}}, \boldsymbol{r})$, where

$$\nabla_{\theta_{k-1}} \hat{\eta}(\pi_{\theta_{k-1}}, \boldsymbol{r}) = \frac{1}{N} \sum_{i=1}^{N} \big((\sum_{t=1}^{T} \nabla_{\theta_{k-1}} \log \pi_{\theta_{k-1}}(a_t^{(i)}|s_t^{(i)}))(\sum_{t=1}^{T} r_t^{(i)}) \big), \tag{4.11}$$

we can derive

$$\nabla_{\theta} \eta_{\pi_{\theta_{k-1}}}(\pi_{\theta_k}) \approx \frac{1}{N} \sum_{i=1}^{N} \big(\Pi_{t=1}^{T} \frac{\pi_{\theta_k}(a_t^{(i)}|s_t^{(i)})}{\pi_{\theta_{k-1}}(a_t^{(i)}|s_t^{(i)})} (\sum_{t=1}^{T} \nabla_{\theta_k} \log \pi_{\theta_k}(a_t^{(i)}|s_t^{(i)}))(\sum_{t=1}^{T} \gamma^{t'-1} r_t^{(i)}) \big). \tag{4.12}$$

and

$$[\nabla_r \theta_k]_t = \sum_{j=1}^{t} \nabla_{\theta_{k-1}} \log \pi_{\theta_{k-1}}(a_j|s_j) \gamma^{t-j} \tag{4.13}$$

Although $\frac{\partial \theta_k}{\partial \check{\boldsymbol{r}}}$ has a closed-form expression for simple learners like VPG, analytically computing how the poisoned reward influences the model is challenging for more complicated learners like PPO, whose update rule is an argmax function. Therefore, we use the Direct Gradient Method proposed by [97] to approximate the gradient by

$$\frac{\partial \eta_{\pi_{\theta_{k-1}}}(\pi_{\theta_k})}{\partial \check{\boldsymbol{r}}} = \frac{\eta_{\pi_{\theta_{k-1}}}(f(\pi_{\theta_{k-1}}, \boldsymbol{r} + \Delta)) - \eta_{\pi_{\theta_{k-1}}}(f(\pi_{\theta_{k-1}}, \boldsymbol{r}))}{\Delta} \tag{4.14}$$

To show the concrete poisoning process, we assume $\mathcal{D} = \mathcal{O}^r$. Then Algorithm 8 shows the detailed procedure of VA2C-P with a non-targeted goal and a white-box attacker.

Algorithm 8: Non-targeted White-box VA2C-P with $\mathcal{O}_{\boldsymbol{r}}$-Poisoning

1 **Input:** total iterations of learning K; poisoning power ϵ; poisoning budget C; attacker's learning rate β; maximum computing iterations J; distribution distance measure d
2 Initialize policy discrepancies as an empty list $\Psi = \emptyset$
3 Initialize the number of already poisoned iterations $c = 0$
4 Initialize value network V_ω
5 **for** $k = 1, \cdots, K$ **do**
6 **if** $c > C$ **then**
7 Break
8 Get the current observation $\mathcal{O}_k$ and the learner's policy model θ_{k-1}
9 Fit the value function: $\omega \leftarrow \text{argmin}_\omega \sum_{\tau_i \in \mathcal{O}_k} \sum_{t=1}^{T} (V_\omega(s_t^{(i)}) - \sum_{t'=t}^{T} \gamma^{t'-t} r_t^{(i)})^2$
10 Imitate learner's update with the clean rewards $\theta_k, \eta \leftarrow \texttt{Update}(\theta_{k-1}, \boldsymbol{r})$
11 Initialize $\check{\boldsymbol{r}}$ as the original $\boldsymbol{r}$ in $\mathcal{O}_k$
12 Set $\eta_0 = \eta_c$
13 **for** $j = 1, \cdots, J$ **do**
14 **for** $i = 1, \cdots, N$ *and* $t = 1, \cdots, T$ **do**
15 Copy $\boldsymbol{r}' \leftarrow \check{\boldsymbol{r}}$, and add a small value Δ to $[r']_t^{(i)}$
16 Imitate learner's update with poisoned rewards $\theta', \eta' \leftarrow \texttt{Update}(\theta_{k-1}, \boldsymbol{r}')$
17 Compute the direct gradient: $\frac{\partial \eta}{\partial \check{\boldsymbol{r}}_t^{(i)}} \leftarrow \frac{\eta' - \eta_{j-1}}{\Delta}$
18 Update the poisoned reward: $\check{\boldsymbol{r}} \leftarrow \Pi_{\mathcal{B}_\epsilon(\boldsymbol{r})}\big(\check{\boldsymbol{r}} - \beta \frac{\partial \eta}{\partial \check{\boldsymbol{r}}}\big)$
19 Imitate learner's update with poisoned rewards $\check{\theta}_k, \eta_j \leftarrow \texttt{Update}(\theta_{k-1}, \boldsymbol{r}')$
20 **if** $(\eta_j - \eta_{j-1})$ *converges* **then**
21 Break
22 Compute $\widehat{\psi}_k = \frac{1}{NT} \sum_{s_t^{(i)}} d(\pi_{\theta_k} || \pi_{\check{\theta}_k})$ and add $\widehat{\psi}_k$ to Ψ
23 **if** $\widehat{\psi}_k$ *is is no lower than the* $(1 - \frac{C-c}{K-k})$*-quantile of* Ψ **then**
24 Attack: replace $\boldsymbol{r}$ with $\check{\boldsymbol{r}}$ in $\mathcal{O}_k$ and send it back to the learner
25 $c \leftarrow c + 1$
26 **Procedure** `Update`$(\theta, \boldsymbol{r})$
27 Perform an update with the learner's algorithm $\theta' \leftarrow f(\theta, \boldsymbol{r})$
28 Compute the attacker's objective
$\eta \leftarrow \frac{1}{NT} \sum_{\tau_i \in \mathcal{O}_k} \sum_{t=1}^{T} (\frac{\pi_{\theta'}(a_t^{(i)} | s_t^{(i)})}{\pi_\theta(a_t^{(i)} | s_t^{(i)})}) (\sum_{t'=t}^{T} \gamma^{t'-t} r_t^{(i)} - V_\omega(s_t^i))$
29 **return** θ', η

For Targeted Poisoning. In line 16, instead of computing $\frac{\partial \eta}{\partial \check{\boldsymbol{r}}_t^{(i)}}$, we compute $\frac{\partial \text{dist}(\theta', \theta^\dagger)}{\partial \check{\boldsymbol{r}}_t^{(i)}}$.

For Black-box Poisoning. In line 7, instead of getting the learner's policy model θ_{k-1}, we train a policy with the same algorithm of the learner $\tilde{\theta}_{k-1} \leftarrow f(\tilde{\theta}_{k-2}, \mathcal{O}_{k-1})$.

4.8.6 Additional Implementation Details and Empirical Results

4.8.6.1 Detailed Experiment Settings

Network Architecture. For all the learners, we use a two-layer policy network with $Tanh$ as the activation function, where each layer has 64 nodes. PPO, A2C and ACKTR also have an additional same-sized critic network. We implement VPG and PPO with PyTorch, and the implementation of A2C and ACKTR are modified from the project by [47].

Hyper-parameters. In all experiments, the discount factor γ is set to be 0.99. We run VPG and PPO for 1000 episodes on every environment, and update the policy after every episode. For A2C and ACKTR, we use 16 processes to collect observations simultaneously, and update policy every 5 steps (i.e., each observation $\mathcal{O}$ has 80 $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$ tuples); learning last for 80000 steps in total.

Distance Measure for Perturbation The definition of total effort function $U(\cdot)$ plays an important role of understanding the attack power. Since states, actions and rewards are in different forms and scales, we define U differently for various poison aims. Also, note that $\mathcal{O}$ is a concatenation of state-action-reward tuples, and its length could vary in different iterations, so we normalize over the length of observation.

For $\mathcal{D} = \mathcal{O}^{\boldsymbol{s}}$, we define the total effort as

$$U(\mathcal{O}^{\boldsymbol{s}}, \check{\mathcal{O}}^{\boldsymbol{s}}) = \frac{1}{\sqrt{|\mathcal{O}^{\boldsymbol{s}}|}} \sum_{s \in \mathcal{O}^{\boldsymbol{s}}} \|s - \check{s}\|_2.$$

For $\mathcal{D} = \mathcal{O}^{\boldsymbol{a}}$, if the action space is continuous, we define the total effort as

$$U(\mathcal{O}^{\boldsymbol{a}}, \check{\mathcal{O}}^{\boldsymbol{a}}) = \frac{1}{\sqrt{|\mathcal{O}^{\boldsymbol{a}}|}} \sum_{a \in \mathcal{O}^{\boldsymbol{a}}} \|a - \check{a}\|_2,$$

and if the action space is discrete, we define the total effort as

$$U(\mathcal{O}^{\boldsymbol{a}}, \check{\mathcal{O}}^{\boldsymbol{a}}) = \frac{1}{\sqrt{|\mathcal{O}^{\boldsymbol{a}}|}} \sum_{a \in \mathcal{O}^{\boldsymbol{a}}} \mathbb{1}(a \neq \check{a}).$$

For example, in CartPole, the action is either 0 or 1, then the attacker with $\epsilon = 0.1$ can flip up to 10% of the actions in one iteration.

For $\mathcal{D} = \mathcal{O}^{\boldsymbol{r}}$, we define the total effort as

$$U(\mathcal{O}^{\boldsymbol{r}}, \check{\mathcal{O}}^{\boldsymbol{r}}) = \frac{1}{\sqrt{|\mathcal{O}^{\boldsymbol{r}}|}} \|\mathbf{r} - \check{\mathbf{r}}\|.$$

In the supplementary materials we provide the code and instructions, as well as demo videos of poisoning A2C in the Hopper environment, where one can see under the same budget constraints, random poisoning has nearly no influence the agent's behaviors, while our proposed VA2C-P successfully prevents the agent from hopping forward.

4.8.6.2 FGSM-based Poisoning

The procedure of the FGSM-based targeted poisoning is as follows. We transfer the method proposed by [31] from attacking DQN to attacking policy-based algorithms. Although Behzadan et al. [31] assume a black-box setting, we hereby use white-box attack (attacker knows learner's policy parameters) in order to let FGSM be a stronger baseline.

For step t,

Step 1: the learner observes s_t takes action a_t, the environment returns reward r_t, state s_{t+1};

Step 2: the attacker queries the target policy and gets $a_{adv} = \pi^{\dagger}(s_{t+1})$;

Step 3: the attacker poisons s_{t+1} by

$$\check{s}_{t+1} = s_{t+1} + \epsilon \times \text{sign}(\nabla_{s_{t+1}}(\pi(a_{adv}|s_{t+1})));$$

Step 4: the attacker sends $\check{s}_{t+1}$ as s_{t+1} to the learner.

4.8.6.3 Additional Experiment Results

Figure 4.6 shows additional results on various environments and RL algorithms, including both non-targeted poisoning and targeted poisoning.

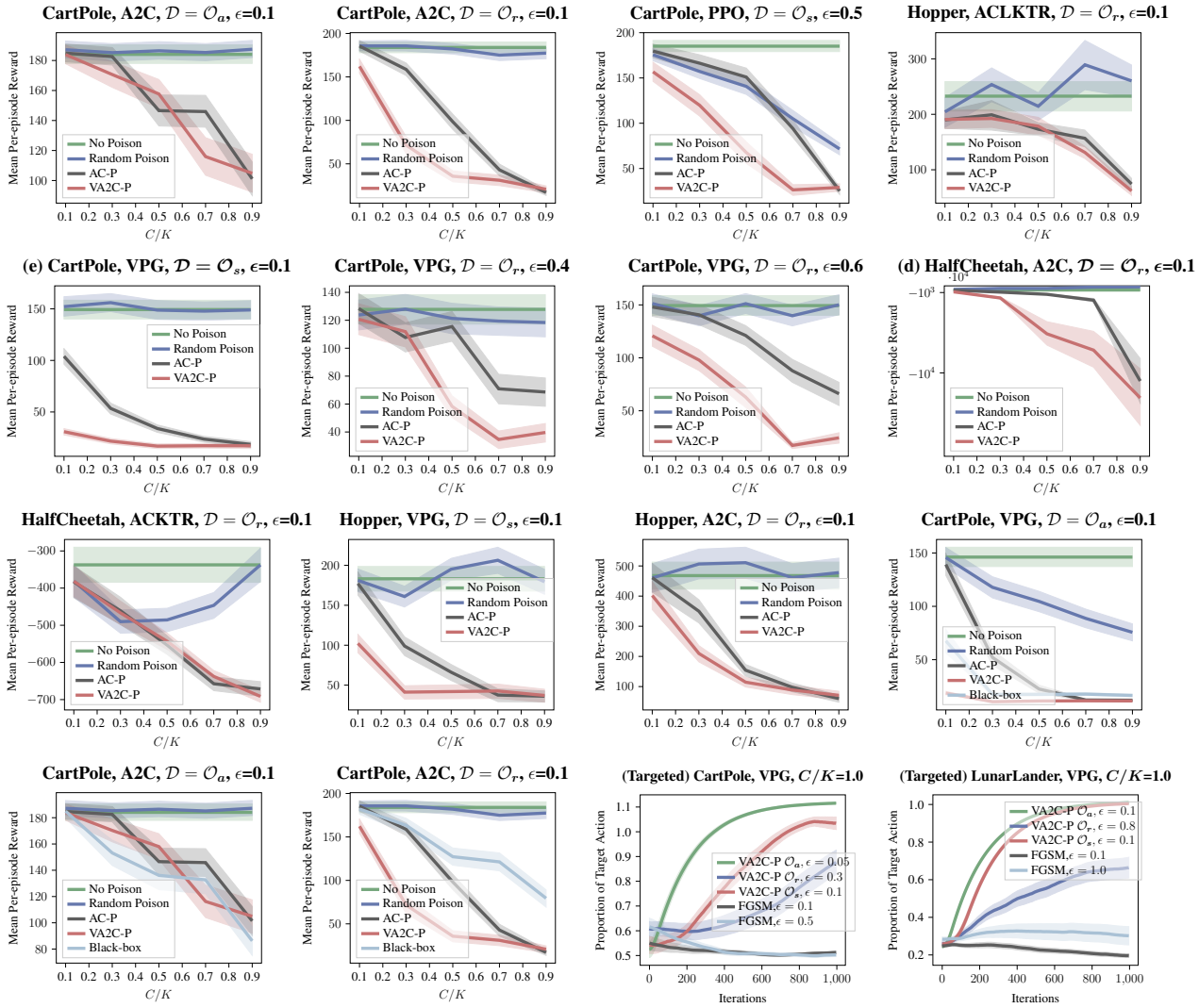


Figure 4.6: Additional Experimental Results.

4.8.6.4 Extension to Off-policy Learners

Although we focus on on-policy policy gradient learners in this chapter, our poisoning method is also applicable to off-policy learners which update their policies using sampled mini-batches from all historical observation (trajectories). If the adversary can manipulate the mini-batch the learner samples at every step, our proposed poisoning process works as usual. We implement this idea and test it for one of the state-of-the-art off-policy learning method SAC [98], and the results are shown in Figure 4.7, where VA2C-P significantly reduces the reward gained by the learner. In the other case, if the adversary doesn't see which mini-batch the learner samples but has access to the buffer, he can still alter or insert some samples to influence learning.

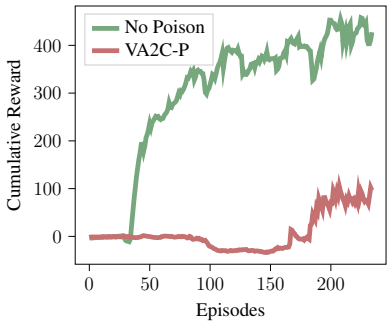


Figure 4.7: Poisoning off-policy algorithm SAC with VA2C-P. $\mathcal{D} = \mathcal{O}^r, C/K = 1, \epsilon = 0.6$.

Chapter 5: Robust Communication in MARL Systems

5.1 Introduction

Neural network-based multi-agent reinforcement learning (MARL) has achieved significant advances in many real-world applications, such as autonomous driving [99, 100]. In a multi-agent system, especially in a cooperative game, communication usually plays an important role. By feeding communication messages as additional inputs to the policy network, each agent can obtain more information about the environment and other agents, and thus can learn a better policy [101, 102, 103]. However, such a communication-dependent policy may not make safe and robust decisions when communication messages are perturbed or corrupted. For example, suppose an agent is trained in a cooperative environment with benign communication, and it learns to trust all communication messages and utilize them. But during test time, there exists a malicious attacker perturbing some communication messages, such that this agent can be drastically misled by the false communication.

The robustness of policy against adversarial communication is crucial for the practical application of MARL. For example, when several drones execute pre-trained policies and exchange information via wireless communication, it is possible that messages get noisy in a hostile environment, or even some malicious attacker eavesdrops on their communication and intentionally perturbs some messages to a victim agent via cyber attacks. Moreover, even if the communi-

(a) Training time

(b) Test time (in deployment)

Figure 5.1: An example of test-time communication attacks in a communicative MARL system. (a) During training, agents are trained collaboratively in a safe environment, such as a simulated environment. (b) In deployment, agents execute pre-trained policies in the real world, where malicious attackers may modify the benign (green) messages into adversarial (red) signals to mislead some victim agent(s).

cation channel is protected by advanced encryption algorithms, an attacker may also hack some agents and alter the messages before they are sent out (e.g. hacking IoT devices that usually lack sufficient protection [104]). Figure 5.1 shows an example of communication attacks, where the agents are trained with benign communication, but attackers may perturb the communication during the test time. The attacker may lure a well-trained agent to a dangerous location through malicious message propagation and cause fatal damage. Although our paper focuses on adversarial perturbations of the communication messages, it also includes unintentional perturbations, such as misinformation due to malfunctioning sensors or communication failures; these natural perturbations are no worse than adversarial attacks.

Achieving high performance in MARL through inter-agent communication while being robust to adversarial communication is a challenging problem due to the following reasons. **Challenge I**: Communication attacks can be stealthy and strong. The attacker may construct a false communication that is far from the original communication, but still semantically meaningful. In the example of Figure 5.1b, the attacker alters "Bomb" to "Gold", which can mislead the victim agent to the location of a bomb. But the victim, without seeing the groundtruth, cannot see the

maliciousness from the message itself. Note that the widely-used ℓ_p threat model [49] does not cover this situation. **Challenge II**: The attacker can even be *adaptive* to the victim agent and significantly reduce the victim's total reward. For instance, for a victim agent who moves according to the average of GPS coordinates sent by others, the attacker may learn to send extreme coordinates to influence the average. **Challenge III**: There can be more than one attacker (or an attacker can perturb more than one message at one step), such that they can collaborate to mislead a victim agent.

Although adversarial attacks and defenses have been extensively studied in supervised learning [68, 105] and reinforcement learning [8, 19], there has been little discussion on the robustness issue against adversarial communication in MARL problems. Some recent works [106, 107, 108] take the first step to investigate adversarial communications in MARL and propose several defending methods. However, these empirical defenses do not fully address the aforementioned challenges, and are not guaranteed to be robust, especially under adaptive attacks. In high-stakes applications, it is also important to ensure robustness with theoretical guarantees and interpretations.

In this chapter, we address all aforementioned challenges by proposing a certifiable defense named **Ablated Message Ensemble (AME)**, that can guarantee the performance of agents when a fraction of communication messages get *arbitrarily* perturbed. Inspired by the ensemble methods which are proved to be the optimal defense against poisoning attacks under the iid sample setting [109], we propose to defend by ablation and ensemble of message sets, which tackles the challenging interactive decision-making under partially observable environments with correlated message samples. The main idea of AME is to make decisions based on multiple different subsets of communication messages. Specifically, for a list of messages coming from different

agents, we train a *message-ablation policy* that takes in a subset of messages and outputs a *base action*. Then, we construct an *message-ensemble policy* by aggregating multiple base actions coming from multiple ablated message subsets. We show that when benign messages are able to reach some consensus, AME aggregates the wisdom of benign messages and thus is resistant to adversarial perturbations, no matter how strong the perturbation is. In other words, AME tolerates arbitrarily strong adversarial perturbations as long as the majority of agents are benign and uniting. Levine et al. [110] use a similar randomized ablation idea to defend against ℓ_0 attacks in image classification. However, they provide high-probability guarantee for classification, which is not suitable for sequential decision-making problems, as the guaranteed probability decreases when it propagates over timesteps.

Summary of Contributions.

(1) We formulate the problem of adversarial attacks and defenses in communicative MARL (CMARL).

(2) We propose a novel defense method, AME, that is certifiably robust against arbitrary perturbations of up to $C < \frac{N-1}{2}$ communications under mild conditions, where N is the number of agents.

(3) Experiment in four multi-agent environments shows that AME obtains significantly higher reward than baseline methods under both non-adaptive and adaptive attackers.

5.2 Related Work

Certifiable Defenses. For more reliable application of deep learning, many approaches have been developed to certify the performance of neural networks, including semidefinite programming-

based defenses [111, 112], convex relaxation of neural networks [66, 69, 70, 71, 72], randomized smoothing of a classifier [57, 113], etc. Most existing works focus on the ℓ_p threat model where the perturbation is small in ℓ_p norm, while we consider a different and practical threat model as discussed in Section 5.3.1.

Adversarial Robustness of RL Agents. To improve the robustness of agents, adversarial training (i.e., introducing adversarial agents to the system during training [8, 18, 36, 114]) and network regularization [19, 39, 50] are empirically shown to be effective under ℓ_p attacks, although such robustness is not theoretically guaranteed. In an effort to certify RL agents' robustness, some approaches [19, 37, 38, 39] apply network certification tools to bound the Q networks. Kumar et al. [55] and Wu et al. [115] apply randomized smoothing [57] to RL for provable robustness.

Communication in MARL. Communication is crucial in solving collaborative MARL problems. There are many existing studies learning communication protocols across multiple agents. Foerster et al. [101] are the first to learn differentiable communication protocols that is end-to-end trainable across agents. Another work by Sukhbaatar et al. [103] proposes an efficient permutation-invariant centralized learning algorithm which learns a large feed-forward neural network to map inputs of all agents to their actions. Attention mechanisms are also proven to be effective in learning communication [116, 117]. Some recent works [118, 119] propose new communication schemes that models other agents' actions or intentions. It is also important to communicate selectively, since some communication may be less informative or unnecessarily expensive. To tackle this challenge, Das et al. [120] propose an attention mechanism for agents to adaptively select which other agents to send messages to. Liu et al. [121] introduce a handshaking procedure so that the agents communicate only when needed.

This chapter proposes a certifiably robust algorithm against perturbations on communications, which is orthogonal to the concrete communication strategy. Our AME can be combined with the above communication MARL methods to improve their robustness.

Adversarial Attacks and Defenses in CMARL. Recently, the existence of adversarial communication in MARL has attracted increasing attention. Blumenkamp et al. [106] shows that in a cooperative game, communication policies of some self-interest agents can hurt other agents' performance. To achieve robust communication, Mitchell et al.[108] adopts a Gaussian Process-based probabilistic model to compute the posterior probabilities that whether each partner is truthful. Tu et al. [122] investigates the vulnerability of multi-agent autonomous systems against ℓ_p communication attacks on vision tasks. Xue et al. [107] proposes to learn an anomaly detector and a message reconstructor to recover the true messages, and train two populations of defenders and attackers to improve the generalizability of defense. But in our formulation, the attacker may arbitrarily replace the messages so that recovering the true message is infeasible.

To the best of our knowledge, our AME is the first certifiable defense in MARL against communication attacks. Moreover, we consider a strong threat model where up to half of the communication messages can be arbitrarily corrupted, capturing many realistic types of attacks.

5.3 Background and Problem Setup

5.3.1 Communicative Muti-agent Reinforcement Learning (CMARL)

Dec-POMDP Model. We consider a Decentralised Partially Observable Markov Decision Process (Dec-POMDP) [120, 123, 124] which is a multi-agent generalization of the single-agent POMDP models. A Dec-POMDP can be modeled as a tuple $\langle \mathcal{D}, \mathcal{S}, \mathcal{A}_{\mathcal{D}}, \mathcal{O}_{\mathcal{D}}, O, P, R, \gamma, \rho_0 \rangle$.

$\mathcal{D} = \{1, \cdots, N\}$ is the set of N agents. $\mathcal{S}$ is the underlying state space. $\mathcal{A}_\mathcal{D} = \times_{i \in \mathcal{D}} \mathcal{A}_i$ is the *joint* action space. $\mathcal{O}_\mathcal{D} = \times_{i \in \mathcal{D}} \mathcal{O}_i$ is the *joint* observation space, with O being the observation emission function. $P : \mathcal{S} \times \mathcal{A}_1 \times \cdots \mathcal{A}_N \to \Delta(\mathcal{S})$ is the state transition function[1]. $R : \mathcal{S} \times \mathcal{A}_1 \times \cdots \mathcal{A}_N \to \mathbb{R}$ is the shared reward function. γ is the shared discount factor, and ρ_0 is the initial state distribution.

Communication Policy ξ in Dec-POMDP. Due to the partial observability, communication among agents is crucial for them to obtain high rewards. Consider a shared message space $\mathcal{M}$, where a message $m \in \mathcal{M}$ can be a scalar or a vector, e.g., signal of GPS coordinates. The communication policy of agent $i \in \mathcal{D}$, denoted as ξ_i, generates messages based on the agent's observation and interaction history. We use $m_{i \to j}$ to denote the message sent from agent i to agent $j \neq i$. In practice, the communication policy can be hand-coded with domain knowledge, or learned with function approximators. Note that our paper focuses on how to defend against adversarial perturbations of existing communication, and *our defense is independent of the underlying communication policy ξ_i.*

Acting Policy π with Communication. The goal of each agent $i \in \mathcal{D}$ is to maximize the discounted cumulative reward $\sum_{t=0}^{\infty} \gamma^t r^{(t)}$ by learning an acting policy π_i. When there exists communication, the policy input contains both its own interaction history, denoted by $\tau_i \in \Gamma_i$, and the communication messages $\mathbf{m}_{:\to i} := \{m_{j \to i} | 1 \leq j \leq N, j \neq i\}$. Similar to the communication policy ξ, we do not make any assumption on how the acting policy π is formulated, as our defense mechanism introduced later can be plugged into any policy learning procedure.

[1] $\Delta(\mathcal{X})$ denotes the space of probability distributions over space $\mathcal{X}$.

5.3.2 Communication Attacks in CMARL

Communication attacks in MARL has recently attracted increasing attention [106, 107, 122] with different focuses, as summarized in Section 5.2. In this chapter, we consider a practical and strong threat model where malicious attackers can arbitrarily perturb a subset of communication messages during test time.

Formulation of Communication Attack: The Threat Model. During test time, agents execute well-trained policies $\pi_1, \cdots, \pi_N$. As shown in Figure 5.1b, the attacker may perturb communication messages to mislead a specific victim agent. Without loss of generality, suppose $i \in \mathcal{D}$ is the victim agent. For notation simplicity, we assume agent i receives messages from all other agents as is common in practice, resulting in $N-1$ communication messages in total. But our defense also works for partically-connected communication graphs. We consider the sparse attack setup where up to C messages can be arbitrarily perturbed at every step, among all $N-1$ messages. Here C is a reflection of the attacker's attacking power. The victim agent has no knowledge of which messages are adversarial. We make the following mild assumption for the attacking power.

Assumption 1 (Attacking Power)**.** *An attacker can* ***arbitrarily*** *manipulate fewer than a half of the communication messages, i.e.,* $C < \frac{N-1}{2}$.

This is a realistic assumption, as it takes the attacker's resources to hack or control each communication channel. It is less likely that an attacker can change the majority of communications among agents without being detected. Note that this is a *very strong threat model* as the C attacked messages can be arbitrarily perturbed based on the attacker's attacking algorithm. Compared to the commonly used ℓ_p attacks, the above threat model can work for broader applications,

such as adding a patch to an image, replacing a word in a sentence, and etc.

We do not make any assumption on what attack algorithm the attacker uses, i.e., how a message is perturbed. That is, we aim to achieve generic and provable robustness under a wide range of practical scenarios. In practice, we do not require the learner to know the exact C, and having an upper bound of C suffices to obtain the guarantees we introduce in the next section.

5.4 Proposed Approach

In this section, we present our defense algorithm, *Ablated Message Ensemble (AME)*, against test-time communication attacks in CMARL. We first introduce the algorithm design in Section 5.4.1, then present the theoretical analysis in Section 5.4.2. Section 5.4.4 discusses an extension of AME.

5.4.1 Ablated Message Ensemble (AME)

Our goal is to learn and execute a robust policy for any agent in the environment, so that the agent can perform well in both a non-adversarial environment and an adversarial environment. To ease the illustration, we focus on robustifying an arbitrary agent $i \in \mathcal{D}$, and the same defense is applicable to all other agents. We omit the agent subscript $_i$, and denote the agent's observation space, action space, and interaction history space as $\mathcal{O}$, $\mathcal{A}$, and Γ, respectively.

Let $\mathbf{m} \in \mathcal{M}^{N-1}$ denote a set of $N-1$ messages received by the agent. Then, we can build an ablated message subset of $\mathbf{m}$ with k randomly selected messages, as defined below.

Definition 38 (***k***-Ablation Message Sample (k-Sample))**.** *For a message set* $\mathbf{m} \in \mathcal{M}^{N-1}$ *and any integer* $1 \leq k \leq N-1$*, define a* k*-ablation message sample (or k-sample for short),* $[\mathbf{m}]_k \in \mathcal{M}^k$*,*

as a set of k randomly sampled messages from $\mathbf{m}$. Let $\mathcal{H}_k(\mathbf{m})$ be the collection of all unique k-samples of $\mathbf{m}$, and thus $|\mathcal{H}_k(\mathbf{m})| = \binom{N-1}{k}$.

We propose Ablated Message Ensemble (AME), a generic defense framework that can be fused with any policy learning algorithm. Motivated by the fact that the benign messages sent from other agents usually contain overlapping information of the environment that may suggest similar decisions, the **main idea** of AME is to make decisions based on the *consensus* of the benign messages. AME has two phases: the training phase where agents are trained in a clean environment, and a testing/defending phase where communications may be perturbed.

Algorithm 9: Training Phase of AME

Input: ablation size k Initialize $\hat{\pi}_i$ for every agent $i \in [N]$. **for** $t = 1, \cdots, \infty$ **do**
  Receive a list of messages $\mathbf{m}_{:\to i}$, get local observation o_i and update interaction history τ_i
  Update the adversarial critic $\tilde{V}_\omega$ with observation $\mathcal{O}_k$
  Randomly sample $[\mathbf{m}_{:\to i}]_k \sim \mathrm{Uniform}(\mathcal{H}_k(\mathbf{m}_{:\to i}))$
  Take action based on τ_i and the k-sample $[\mathbf{m}_{:\to i}]_k$, i.e., $a_i \leftarrow \hat{\pi}_i(\tau_i, [\mathbf{m}_{:\to i}]_k)$ Update the replay buffer and policy $\hat{\pi}_i$ **Output:** message-ablation policy $\hat{\pi}_i, \forall i \in [N]$

Algorithm 10: Defending Phase of AME

Input: ablation size k Initialize $\hat{\pi}_i$ for every agent $i \in [N]$. **for** $t = 1, \cdots, \infty$ **do**
  Receive a list of messages $\mathbf{m}_{:\to i}$, get local observation o_i and update interaction history τ_i
  Update the adversarial critic $\tilde{V}_\omega$ with observation $\mathcal{O}_k$
  Randomly sample $[\mathbf{m}_{:\to i}]_k \sim \mathrm{Uniform}(\mathcal{H}_k(\mathbf{m}_{:\to i}))$
  Take action based on τ_i and the k-sample $[\mathbf{m}_{:\to i}]_k$, i.e., $a_i \leftarrow \hat{\pi}_i(\tau_i, [\mathbf{m}_{:\to i}]_k)$ Update the replay buffer and policy $\hat{\pi}_i$ **Output:** message-ablation policy $\hat{\pi}_i, \forall i \in [N]$

Training Phase with Message-Ablation Policy $\hat{\pi}$ (Algorithm 9). During training time, the agent learns a *message-ablation policy* $\hat{\pi} : \Gamma \times \mathcal{M}^k \to \mathcal{A}$ which maps its own interaction history τ and a random k-sample $[\mathbf{m}]_k \sim \mathrm{Uniform}(\mathcal{H}_k(\mathbf{m}))$ to an action, where $\mathrm{Uniform}(\mathcal{H}_k(\mathbf{m}))$ is the uniform distribution over all k-samples from the message set $\mathbf{m}$ it receives. Here k is a user-

specified hyperparameter selected to satisfy certain conditions, as discussed in Section 5.4.2. The training objective is to maximize the cumulative reward of $\hat{\pi}$ based on randomly sampled k-samples in a non-adversarial environment. Any policy optimization algorithm can be used for training. Although by ablating a subset of messages, the message-ablation policy may compromise some natural reward (which is typical for robust training methods), it can significantly enhance the robustness of agents when attacks exist, with our design of test-time defense mechanism introduced below.

Defending Phase with Message-Ensemble Policy $\widetilde{\pi}$ (Algorithm 10). Once we obtain a reasonable message-ablation policy $\hat{\pi}$, we can execute it with ablation and aggregation during test time to defend against adversarial communication. The main idea is to collect all possible k-samples from $\mathcal{H}_k(\mathbf{m})$, and select an action voted by the majority of those k-samples. Specifically, we construct a *message-ensemble policy* $\widetilde{\pi} : \Gamma \times \mathcal{M}^{N-1} \to \mathcal{A}$ that outputs an action by aggregating the base actions produced by $\hat{\pi}$ on multiple k-samples (Line 5 in Algorithm 10). The construction of the message-ensemble policy depends on whether the agent's action space $\mathcal{A}$ is discrete or continuous, which is given below by Definition 39.

Definition 39 (Message-Ensemble Policy)**.** *For a message-ablation policy $\hat{\pi}$ with observation history τ and received message set $\mathbf{m}$, define the message-ensemble policy $\widetilde{\pi}$ as*

$$\widetilde{\pi}(\tau, \mathbf{m}) := \operatorname{argmax}_{a \in \mathcal{A}} \sum_{[\mathbf{m}]_k \in \mathcal{H}_k(\mathbf{m})} \mathbb{1}[\hat{\pi}(\tau, [\mathbf{m}]_k) = a], \qquad \textit{if } \mathcal{A} \textit{ is discrete, and} \tag{5.1}$$

$$\widetilde{\pi}(\tau, \mathbf{m}) := \mathsf{Median}\{\hat{\pi}(\tau, [\mathbf{m}]_k) : [\mathbf{m}]_k \in \mathcal{H}_k(\mathbf{m})\}, \qquad \textit{if } \mathcal{A} \textit{ is continuous.} \tag{5.2}$$

Therefore, the message-ensemble policy $\widetilde{\pi}$ takes the action suggested by the consensus of all k-samples (majority vote in a discrete action space, and coordinate-wise median for a continuous action space). Different from model-ensemble methods [125, 126] that learn multiple

network models, we use the ensemble of messages, and only train a single policy network $\hat{\pi}$. Therefore, the *training process does not require extra computations*. We will show in the next section that, with some mild conditions, $\widetilde{\pi}$ under adversarial communications works similarly as the message-ablation policy $\hat{\pi}$ under all-benign communications.

5.4.2 Robustness Certificates of AME

In this section, we theoretically analyze the robustness of AME. During test time, at any step, let τ be the interaction history, $\mathbf{m}_{\text{benign}}$ be the unperturbed benign message set, and $\mathbf{m}_{\text{adv}}$ be the perturbed message set. Note that $\mathbf{m}_{\text{benign}}$ and $\mathbf{m}_{\text{adv}}$ both have $N-1$ messages while they differ by up to C messages. With the above notation, we define a set of actions rendered by purely benign k-samples in Definition 40. As the agent using a well-trained message-ablation policy is likely to take these actions in a non-adversarial environment, they can be regarded as "good" actions to take.

Definition 40 (Benign Action Set $\mathcal{A}_{\text{benign}}$)**.** *For the execution of the message-ablation policy $\hat{\pi}$ at any step, define $\mathcal{A}_{\text{benign}} \subseteq \mathcal{A}$ as a set of actions that $\hat{\pi}$ may select under benign k-samples.*

$$\mathcal{A}_{\text{benign}} := \cup_{[\mathbf{m}_{\text{benign}}]_k \in \mathcal{H}_k(\mathbf{m}_{\text{benign}})} \left\{\hat{\pi}(\tau, [\mathbf{m}_{\text{benign}}]_k)\right\}. \tag{5.3}$$

Our robustness certificates are based on the intuition that the consensus of ablated messages is benign, under the condition that a consensus exists among benign messages. Next, we present the action certificates and reward certificates for discrete and continuous actions, respectively.

5.4.2.1 Robustness Certificates for Discrete Action Space

We first characterize the condition for the existence of consensus. Intuitively, the following condition ensures that the action resulted by majority vote stands for the consensus of benign messages.

Condition 1 (Dominating Benign Votes)**.** *Denote the highest number of votes among all actions given message set* $\mathbf{m}$ *as* $u_{\max}(\mathbf{m}) := \max_{a\in\mathcal{A}} \sum_{[\mathbf{m}]_k\in\mathcal{H}_k(\mathbf{m})} \mathbb{1}[\hat{\pi}(\tau,[\mathbf{m}]_k)=a]$ *satisfies*

$$u_{\max}(\mathbf{m}) > \binom{N-1}{k} - \binom{N-1-C}{k} =: u_{\text{adv}}, \tag{5.4}$$

where u_{adv} *is the number of votes that adversarial messages may affect (the number of k-samples that contain at least one adversarial message).*

Remarks. (1) This condition ensures the consensus has more votes than the votes that adversarial messages are involved in. Therefore, when $\widetilde{\pi}$ takes an action, there must exist some purely-benign k-samples voting for this action. (2) This condition is easy to satisfy when $C \ll N$ as $\binom{N-1}{k} \approx \binom{N-1-C}{k}$. (3) This condition can be easily checked during execution given an upper bound of C.

The following theorem suggests that under the above condition. the ensemble policy $\widetilde{\pi}$ always takes benign actions suggested by benign message combinations, no matter whether attacks exist.

Theorem 41 (Action Certificate for Discrete Action Space)**.** *For a perturbed message set* $\mathbf{m}_{\text{adv}}$*, if Condition 1 holds, the ensemble policy* $\widetilde{\pi}$ *in Equation* (5.1) *produces benign actions under* $\mathbf{m}_{\text{adv}}$*:*

$$\widetilde{a} = \widetilde{\pi}(\tau, \mathbf{m}_{\text{adv}}) \in \mathcal{A}_{\text{benign}}. \tag{5.5}$$

Then, we can further derive a reward certificate as introduced below.

Reward Certificate Theorem 41 justifies that under sufficient majority votes, the message-ensemble policy $\widetilde{\pi}$ ignores the malicious messages in $\mathbf{m}_{\text{adv}}$ and executes a benign action that is suggested by some benign message combinations, even when the malicious messages are not identified. It is important to note that, when the message-ensemble policy $\widetilde{\pi}$ selects an action $\widetilde{a}$, there must exist at least one purely benign k-sample that let the message-ablation policy $\hat{\pi}$ produce $\widetilde{a}$. Therefore, as long as $\hat{\pi}$ can obtain high reward with randomly selected benign k-samples, $\widetilde{\pi}$ can also obtain high reward with ablated adversarial communication due to its design.

Specifically, we consider a specific agent with message-ablation policy $\hat{\pi}$ and message-ensemble policy $\widetilde{\pi}$ (suppose other agents are executing fixed policies). Let $\nu : \mathcal{M}^{N-1} \to \mathcal{M}^{N-1}$ be an attack algorithm that perturbs at most C messages in a message set. Let $\zeta \sim Z(P, \hat{\pi})$ be a trajectory of policy $\hat{\pi}$ under no attack, i.e., $\zeta = (o^{(0)}, \mathbf{m}^{(0)}, a^{(0)}, r^{(0)}, o^{(1)}, \mathbf{m}^{(1)}, a^{(0)}, r^{(0)}, \cdots)$. (Recall that a message-ablation policy $\hat{\pi}$ takes in a random size-k subset of $\mathbf{m}^{(t)}$ and outputs action $a^{(t)}$.) When there exists attack with ν, let $\zeta_\nu \sim Z(P, \widetilde{\pi}; \nu)$ be a trajectory of policy $\widetilde{\pi}$ under communication attacks, i.e., $\zeta_\nu = (o^{(0)}, \nu(\mathbf{m}^{(0)}), a^{(0)}, r^{(0)}, o^{(1)}, \nu(\mathbf{m}^{(1)}), a^{(0)}, r^{(0)}, \cdots)$. For any trajectory ζ, let $r(\zeta)$ be the discounted cumulative reward of this trajectory.

With the above notations, we propose the following reward certificate.

Corollary 42 (Reward Certificate for Discrete Action Space)**.** *When Condition 1 holds at every step of execution, the cumulative reward of ensemble policy $\widetilde{\pi}$ defined in Equation (5.1) under adversarial communication is no lower than the lowest cumulative reward that the ablation policy $\hat{\pi}$ can obtain with randomly selected k-samples under no attacks, i.e.,*

$$\min_{\zeta_\nu \sim Z(P, \widetilde{\pi}; \nu)} r(\zeta_\nu) \geq \min_{\zeta \sim Z(P, \hat{\pi})} r(\zeta), \tag{5.6}$$

for any attacker ν satisfying Assumption 1 ($C < \frac{N-1}{2}$).

Remarks. (1) The certificate holds for any attack algorithm ν with $C < \frac{N-1}{2}$. (2) The message-ablation policy $\hat{\pi}$ has extra randomness from the sampling of k-samples. That is, at every step, $\hat{\pi}$ takes a uniformly randomly selected k-sample from $\mathcal{H}_k(\mathbf{m})$. Therefore, the $\min_\zeta$ in the RHS considers the worst-case message sampling in the clean environment without attacks. Since $\widetilde{\pi}$ always takes actions selected by some purely-benign message combinations, the trajectory generated by $\widetilde{\pi}$ can also be produced by the message-ablation policy. (3) Note that the RHS ($\min_{\zeta \sim Z(P,\hat{\pi})} r(\zeta)$) can be approximately estimated by executing $\hat{\pi}$ during training time, so that the test-time performance of $\widetilde{\pi}$ is guaranteed to be no lower than this value, even if there are up to C corrupted messages at every step.

5.4.2.2 Robustness Certificates for Continuous Action Space

For a continuous action space, the following condition is needed, which can always be satisfied by the proper selection of ablation size k.

Condition 2 (Dominating Benign Samples)**.** *The ablation size k of AME satisfies*

$$\binom{N-1-C}{k} > \frac{1}{2}\binom{N-1}{k}. \tag{5.7}$$

Remarks. (1) For the message set $\mathbf{m}_{\text{adv}}$ that has up to C adversarial messages, this condition implies that among all k-samples from $\mathbf{m}_{\text{adv}}$, there are more purely benign k-samples than k-samples that contain adversarial messages. (2) Under Assumption 1, this condition *always has solutions* for k. (3) With an upper bound of C, one can choose k as the largest solution to Equation (5.7).

Theorem 43 (Action Certificate for Continuous Action Space)**.** *Under Condition 2, the action* $\widetilde{a} = \widetilde{\pi}(\tau, \mathbf{m}_{\text{adv}})$ *generated by the ensemble policy* $\widetilde{\pi}$ *defined in Equation* (5.2) *satisfies*

$$\widetilde{a} \in \mathsf{Range}(\mathcal{A}_{\text{benign}}) := \{a : \forall 1 \leq l \leq L, \exists \underline{a}, \overline{a} \in \mathcal{A}_{\text{benign}} \; s.t \; \underline{a}_l \leq a_l \leq \overline{a}_l\}. \tag{5.8}$$

Theoretically, $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ is a set of actions that are coordinate-wise bounded by base actions resulted from purely benign k-samples. In many practical problems, it is reasonable to assume that actions in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ are relatively safe, especially when benign actions in $\mathcal{A}_{\text{benign}}$ are concentrated. For instance, if benign message combinations have suggested driving at 40 mph or driving at 50 mph, then driving at 45 mph is also safe.

Note that Theorem 43 certifies that the selected action is in a set of actions that are close to benign actions $\mathsf{Range}(\mathcal{A}_{\text{benign}})$, but does not make any assumption on this set. Next we interpret this result in details.

How to Understand $\mathsf{Range}(\mathcal{A}_{\text{benign}})$**?** Theoretically, $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ is a set of actions that are coordinate-wise bounded by base actions resulted from purely benign k-samples. In many practical problems, it is reasonable to assume that actions in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ are relatively safe, especially when benign actions in $\mathcal{A}_{\text{benign}}$ are concentrated. The following examples illustrate some scenarios where actions in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ are relatively good.

1. If the action denotes the price a seller sells its product in a market, and the communication messages are the transaction signals in an information pool, then $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ is a price range that is determined based on purely benign messages. Therefore, the seller will set a reasonable price without being influenced by a malicious message.

2. If the action denotes the driving speed, and benign message combinations have suggested driving at 40 mph or driving at 50 mph, then driving at 45 mph is also safe.

3. If the action is a vector denoting movements of all joints of a robot (as in many MuJoCo tasks), and two slightly different joint movements are suggested by two benign message combinations, then an action that does not exceed the range of the two benign movements at every joint is likely to be safe as well.

The above examples show by intuition why the message-ensemble policy can be regarded as a relatively robust policy. However, in extreme cases where there exists "caveat" in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$, taking an action in this set may also be unsafe. To quantify the influence of $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ on the long-term reward, we next analyze the cumulative reward of the message-ensemble policy in the continuous-action case.

How Does $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ Lead to A Reward Certificate? Different from the discrete-action case, the message-ensemble policy $\widetilde{\pi}$ in a continuous action space may take actions not in $\mathcal{A}_{\text{benign}}$ such that it generates trajectories not seen by the message-ablation policy $\hat{\pi}$. However, since the action of $\widetilde{\pi}$ is guaranteed to stay in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$, we can bound the difference between the value of $\widetilde{\pi}$ and the value of $\hat{\pi}$, and how large the different is depends on some properties of $\mathsf{Range}(\mathcal{A}_{\text{benign}})$.

Concretely, Let R and P be the reward function and transition probability function of the current agent when the other agents execute fixed policies. So $R(s,a)$ is the immediate reward of taking action a at state s, and $P(s'|s,a)$ is the probability of transitioning to state s' from s by taking action a. (Note that s is the underlying state which may not be observed by the agent.)

Definition 44 (Dynamics Discrepancy of $\hat{\pi}$)**.** *A message-ablation policy $\hat{\pi}$ is called ϵ_R,ϵ_P-discrepant if ϵ_R, ϵ_P are the smallest values such that for any $s \in \mathcal{S}$ and the corresponding benign action set*

$\mathcal{A}_{\text{benign}}$, *we have* $\forall a_1, a_2 \in \mathsf{Range}(\mathcal{A}_{\text{benign}})$,

$$|R(s, a_1) - R(s, a_2)| \leq \epsilon_R, \tag{5.9}$$

$$\int |P(s'|s, a_1) - P(s'|s, a_2)| \mathrm{d}s' \leq \epsilon_P. \tag{5.10}$$

Remarks. (1) Equation (5.10) is equivalent to that the total variance distance between $P(\cdot|s, a_1)$ and $P(\cdot|s, a_2)$ is less than or equal to $\epsilon_P/2$. (2) For any environment with bounded reward, ϵ_R and ϵ_P always exist.

Definition 44 characterizes how different the local dynamics of actions in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ are, over all possible states. If $\mathsf{Range}(\mathcal{A}_{\text{benign}})$ is small and the environment is relatively smooth, then taking different actions within this range will not result in very different future rewards. The theorem below shows a reward certificate for the message-ensemble policy $\widetilde{\pi}$.

Theorem 45 (Reward Certificate for Continuous Action Space)**.** *Let $V^{\hat{\pi}}(s)$ be the clean value (discounted cumulative reward) of $\hat{\pi}$ starting from state s under no attack; let $\tilde{V}_{\nu}^{\widetilde{\pi}}(s)$ be the value of $\widetilde{\pi}$ starting from state s under attack algorithm ν, where ν satisfies Assumption 1; let k be an ablation size satisfying Condition 2. If $\hat{\pi}$ is ϵ_R,ϵ_P-discrepant, then for any state $s \in \mathcal{S}$, we have*

$$\min_{\nu} \tilde{V}_{\nu}^{\widetilde{\pi}}(s) \geq V^{\hat{\pi}}(s) - \frac{\epsilon_R + \gamma V_{\max} \epsilon_P}{1 - \gamma}, \tag{5.11}$$

where $V_{\max} := \sup_{s,\pi} |V^{\pi}(s)|$.

Remarks. (1) The certificate holds for any attack algorithm ν with $C < \frac{N-1}{2}$. (2) If ϵ_R and ϵ_P are small, then the performance of message-ensemble policy $\widetilde{\pi}$ under attacks is similar to the performance of the message-ablation policy $\hat{\pi}$ under no attack.

It is important to note that ϵ_R and ϵ_P are intrinsic properties of $\hat{\pi}$, independent of the attacker. Therefore, one can approximately measure ϵ_R and ϵ_P during training. Similar to Condition 1 required for a discrete action space, the gap between the attacked reward of $\widetilde{\pi}$ and the

natural reward of $\hat{\pi}$ depends on how well the benign messages are reaching a consensus. (Smaller ϵ_R and ϵ_P imply that the actions in $\mathcal{A}_{\text{benign}}$ are relatively concentrated and the environment dynamics are relatively smooth.)

Moreover, one can optimize $\hat{\pi}$ during training such that ϵ_R and ϵ_P are as small as possible, to further improve the robustness guarantee of $\widetilde{\pi}$. This can be a future extension of this work.

5.4.3 Interpretation of Conditions and Hyperparameter

How to Select Ablation Size k: Trade-off between Performance and Robustness. The ablation size k is an important hyperparameter for the guarantees to hold. In general, a smaller k can tolerate a larger C, and a smaller C can be defended by a wider range of k. However, a smaller k restricts the power of information sharing, as the message-ablation policy can access fewer messages in one step. Therefore, the value of k is related to the intrinsic trade-off between robustness and natural performance [105, 127], which suggests that seeking high natural performance may hurt robustness, and *achieving strong robustness may sacrifice some natural performance*. In practice, one can also train several message-ablation policies with different k's during training, and later adaptively select a message-ablation policy to construct a message-ensemble policy during execution (decrease k if higher robustness is needed).

Worst-Case Adversaries. Condition 1 and Condition 2 consider the worst-case scenario when the adversarial messages collaborate to dominate the consensus, which is intrinsically harsh for the victim — if the victim fails to identify or filter out the adversarial messages, its performance can be arbitrarily bad in the worst case. In contrast, although AME sacrifices some natural performance, its worst-case performance can be guaranteed. When these conditions do not hold,

our algorithm can still achieve strong robustness under sub-optimal attacks, as verified in experiments.

Information Redundancy Leads to Higher Robustness. Intuitively, AME prefers benign messages to contain certain amount of redundant information (consensus), which is similar to a multi-factor authentication in security-critical applications. This allows AME to tolerate arbitrary perturbations without assumptions such as a bounded ℓ_p-norm. More importantly, AME provides a way to establish and utilize such consensus, by explicitly setting k and checking the required conditions.

5.4.4 Scaling Up: Ensemble with Partial Samples

So far we have discussed the proposed AME defense and the constructed ensemble policy that aggregates all $\binom{N-1}{k}$ number of k-samples out of $N-1$ messages. However, if N is large, sampling all $\binom{N-1}{k}$ combinations of message subsets could be expensive. In this case, a smaller number of k-samples can be used. That is, given a sample size $0 < D \leq \binom{N-1}{k}$, we randomly select D number of k-samples from $\mathcal{H}_k(\mathbf{m})$ (without replacement), and then we aggregate the message-ablation policy's decisions on selected k-samples. In this way, we define a partial-sample version of the ensemble policy, namely *D-ensemble policy* $\widetilde{\pi}_D$.

Formally, let $\mathcal{H}_{k,D}(\mathbf{m})$ be a subset of $\mathcal{H}_k(\mathbf{m})$ that contains D random k-samples from $\mathcal{H}_k(\mathbf{m})$. Then the D-ensemble policy π_D is defined as

$$\widetilde{\pi}_D(\tau, \mathbf{m}) := \text{argmax}_{a \in \mathcal{A}} \sum_{[\mathbf{m}]_k \in \mathcal{H}_{k,D}(\mathbf{m})} \mathbb{1}[\hat{\pi}(o, [\mathbf{m}]_k) = a], \tag{5.12}$$

for a discrete action space, and

$$\tilde{\pi}_D(\tau, \mathbf{m}) = \mathsf{Median}\{\hat{\pi}(\tau, [\mathbf{m}]_k)\}_{[\mathbf{m}]_k \in \mathcal{H}_{k,D}(\mathbf{m})}. \tag{5.13}$$

for a continuous action space.

In the partial-sample version of AME, we can still provide high-probability robustness guarantees.

For notation simplicity, let $n_1 = \binom{N-1}{k}$, $n_2 = \binom{N-C-1}{k}$. Define the majority vote as

$$u_{\max}(\mathbf{m}) := \max_{a \in \mathcal{A}} \sum\nolimits_{[\mathbf{m}]_k \in \mathcal{H}_{k,D}(\mathbf{m})} \mathbb{1}[\hat{\pi}(\tau, [\mathbf{m}]_k) = a]. \tag{5.14}$$

The following theorem shows a general guarantee for D-ensemble policy in a discrete action space.

Theorem 46 (General Action Guarantee for Discrete Action Space)**.** *Given an arbitrary sample size $0 < D \leq \binom{N-1}{k}$, for the D-ensemble policy $\widetilde{\pi}_D$ defined in Equation (5.12), Equation (5.5) holds deterministically if the majority vote $u_{\max}(\mathbf{m}_{\text{adv}}) > n_1 - n_2$. Otherwise it holds with probability at least*

$$p_D = \frac{\sum_{j=0}^{u_{\max}(\mathbf{m}_{\text{adv}})-1} \binom{n_1-n_2}{j}\binom{n_2}{D-j}}{\binom{n_1}{D}}. \tag{5.15}$$

Note that Theorem 41 is a special case of Theorem 46, since it assumes $u_{\max}(\mathbf{m}_{\text{adv}}) > n_1 - n_2$.

Theorem 47 below further shows the theoretical result for a continuous action space.

Theorem 47 (General Action Guarantee for Continuous Action Space)**.** *Given an arbitrary sample size $0 < D \leq \binom{N-1}{k}$, for the D-ensemble policy $\widetilde{\pi}_D$ defined in Equation (5.13) with an ablation size k satisfying Condition 2, Equation (5.8) holds with probability at least*

$$p_D = \frac{\sum_{j=\tilde{D}}^{D} \binom{n_2}{j}\binom{n_1-n_2}{D-j}}{\binom{n_1}{D}}, \tag{5.16}$$

where $\tilde{D} = \lfloor \frac{D}{2} \rfloor + 1$.

The larger D is, the higher the probability p_D is, the more likely that the message-ensemble policy selects an action in $\mathsf{Range}(\mathcal{A}_{\text{benign}})$. In Theorem 47, when $D = \binom{N-1}{k}$, the probability p_D is 1 and the result matches Theorem 43.

Technical proofs of all theoretical results can be found in Section 5.7.1.

5.5 Empirical Results

In this section, we verify the robustness of our AME in multiple different CMARL environments against various communication attack algorithms. Then, we conduct hyperparameter tests for the ablation size k and the sample size D (for the variant of AME introduced in Section 5.4.4).

Environments. To evaluate the effectiveness of AME, we consider the following four environments that cover various environment and communication settings.

• *FoodCollector (discrete/continuous action, pre-defined communication)*: $N = 9$ agents with different colors search for foods with the same colors. Agents use their limited-range sensors to observe the surrounding objects, while communicating with each other to share their recently observed food locations. The action space can be either discrete (9 moving directions and 1 no-move action), or continuous (2-dimensional vector denoting acceleration). Agents are penalized for having leftover foods at every step, so they seek to find all foods as fast as possible.

• *InventoryManager (continuous action, pre-defined communication)*: $N = 10$ cooperative distributor agents carry inventory for 3 products. There are 300 buyers sharing an underlying demand distribution. At every step, each buyer requests products from a randomly selected distributor. Then, each distributor agent observes random demand requests, takes restocking actions

to adjust its own inventory, and communicates with others by sending its demand observations. Distributors are penalized for mismatches between their inventory and the demands.

• *MARL-MNIST (discrete action, learned communication)*[128]: $N = 9$ agents aim to classify images in MNIST within a limited time horizon. At every step, each agent observes a patch of the image, and can take an action to move to an adjacent patch. Agents are allowed to exchange information (encoded local beliefs) with other agents to update their own beliefs.

• *Traffic Junction (discrete action, learned communication) [103]*: $N = 10$ cars move along potentially intersecting routes on one or more road junctions. Each car has a limited visibility but is free to communicate with all other cars. The goal is pass the road without collision.

Baselines. Based on the same policy learning paradigm for each environment, we compare our AME with two defense baselines: **(1)** *(Vanilla)*: vanilla training without defense, which learns a policy based on all benign messages. **(2)** *(AT)* adversarial training as in [18], which alternately trains an adaptive RL attacker and an agent. As is common in related work [129, 130], we use parameter sharing and train a shared policy network for all agents. Due to the symmetricity of the agents, we train AME and baselines using PPO [16] with parameter sharing [131]. For MNIST-Classification, our implementation is based on [132] that reproduces the results of [128]. During training and defending, we set the ablation size AME as $k = 2$, the largest solution to Equation (5.7) for $C = 2$. For AT, we train the agent against $C = 2$ learned adversaries. More implementation details are provided in Section 5.7.4.1.

Evaluation Metrics. To evaluate the effectiveness of our defense strategy, we test the performance of the trained policies under no attack and under various values of C (for simplicity, we refer to C as the *number of adversaries*). Different from previous work [106] where the adversarial agent disrupts all the other agents, we consider the case where the attacker deliberately

misleads a selected victim, which could evaluate the robustness of the victim under the harshest attacks. We report the victim's local reward under the following two types of attack methods:

(1) *Heuristic attack* that perturbs messages based on heuristics. In FoodCollector, MARL-MNIST and Traffic Junction, we consider randomly generated messages within the valid range of communication messages. Note that this is already a strong attack since a random message could be arbitrarily far from the original message. For InventoryManager where messages have clear physical meanings (demand of buyers), we consider 3 realistic attacks: Perm-Attack, Swap-Attack, Flip-Attack, which permute, swap or flip the demand observations, respectively, as detailed in Section 5.7.4.1.

(2) *Learned adaptive attack* that learns the strongest/worst adversarial communication with an RL algorithm to minimize the victim's reward (a white-box attacker which knows the victim's reward). The learned attack can strategically mislead the victim. As shown in prior works [8, 18,

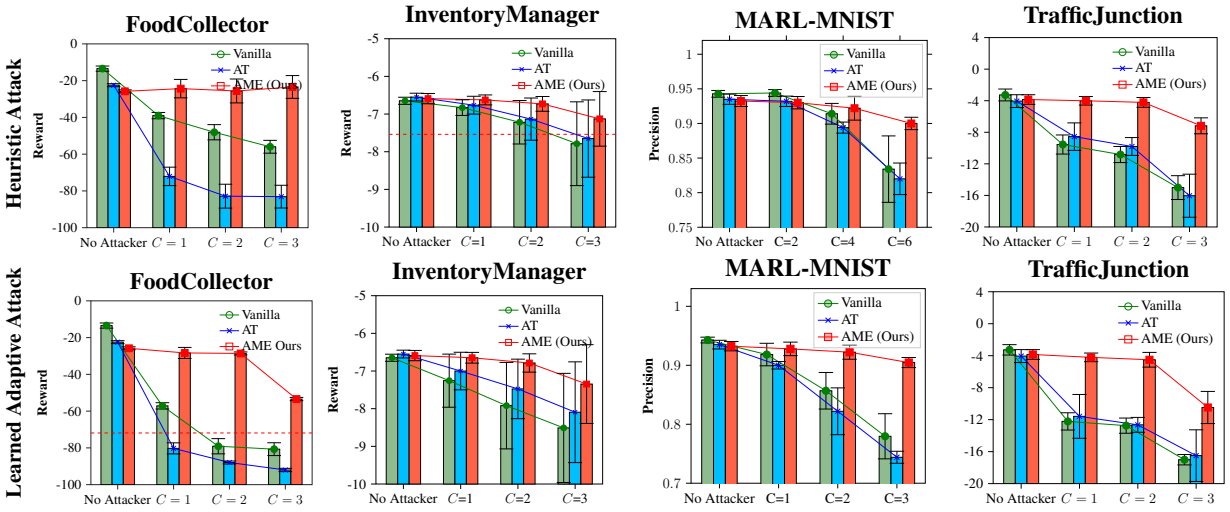


Figure 5.2: Rewards of our AME and baselines in all environments, under no attacker and varying numbers of adversaries for adaptive and various non-adaptive attacks. The dashed red lines stand for the average performance of a non-communicative agent. Results are averaged over 5 random seeds. Our AME outperforms all baseline methods in all tasks, and stays robust for varying number of adversaries (denoted by C).

19], the theoretically optimal attack (which minimizes the victim's reward) can be formed as an RL problem and learned by RL algorithms. Therefore, we can regard this attack as a worst-case attack for the victim agents. More details are in Section 5.7.4.1.

Experiment Results. The major results are shown in Figure 5.2, where we present the discrete-action FoodCollector, and use Swap-Attack as the heuristic attack for InventoryManager. More results including continuous-action FoodCollector and other heuristic attacks for InventoryManager are put in Section 5.7.4.2. We can see that the rewards of Vanilla and AT drastically drop under attacks. Under strong adaptive attackers, Vanilla and AT sometimes perform worse than a non-communicative agent shown by dashed red lines, which suggests that communication can be a double-edged sword. Although AT is usually effective for ℓ_p attacks [18], we find that AT does not achieve better robustness than Vanilla, since it can not adapt to arbitrary perturbations to several messages. In contrast, *AME can utilize benign communication well while being robust to adversarial communication.*

We use $k = 2$ for our AME, which in theory provides performance guarantees against up to $C = 2$ adversaries for $N = 9$ and $N = 10$. We can see that the reward of AME under $C = 1$ or $C = 2$ is similar to its reward under no attack, matching our theoretical analysis. Even under 3 adversaries where the theoretical guarantees no longer hold, AME still obtains superior performance compared to Vanilla and AT. Therefore, *AME makes agents robust under varying numbers of adversaries.*

Discussion on Ablation Size k We demonstrate AME's natural reward and attacked reward in discrete-action FoodCollector under $C = 2$ adversaries with *all possible values of* k *ranging from* 1 *to* $N - 1$ in Figure 5.3(left). The green curve shows that the natural performance of AME increases as k gets larger, where $k = 1$ is the most conservative version of AME, and

Figure 5.3: Natural and robust performance of AME with various values of **(left)** ablation size k and **(right)** sample size D, in discrete-action FoodCollector under $C = 2$.

$k = N - 1$ degenerates to the vanilla policy without message ablation. It is intuitive because a larger k allows the agent to gather more information from others. Although a larger k is beneficial in a clean environment, gathering information without defense makes the agent more vulnerable to communication attacks. The red and blue curves show that the reward of agents decreases under attacks when k gets larger, especially when attacks are adaptive. Therefore, *increasing ablation size k trades off robustness for natural performance*, matching the analysis in Section 5.4.3. As the largest solution to Equation (5.7) in Condition 2, ablation size $k = 2$ achieves a good balance between performance and robustness. Even when $k > 2$ which breaks Condition 2, AME is still more robust than baselines, showing the flexibility of AME.

Discussion on for Sample Size D We evaluate the partial-sample variant of AME introduced in Section 5.4.4 with D varying from $\binom{N-1}{k}$ (ensemble of all message combinations) to 1 (randomly take one k-sample), under $k = 2$ and $C = 2$. Figure 5.3(right) demonstrates the performance of different D's in discrete-action FoodCollector, and Section 5.7.4.2 shows results in other environments. As D goes down, AME obtains lower reward under attackers, but it is still significantly more robust than baseline methods. Note that $D = 1$ is equivalent to executing the message-ablation policy without ensemble, which is robust to heuristic attacks but less robust to

adaptive attacks than the original AME, verifying the effectiveness of message ensemble.

5.6 Conclusion

This chapter proposes a defense framework AME, which is certifiably robust against multiple arbitrarily perturbed adversarial communication messages, for any existing communication protocol, based on randomized ablation and aggregation of messages. Our proposed ablation and ensemble method can be extended to robustify other decision makers which takes in multiple possibly-untrustworthy information sources. The limitation of AME is the requirement of several conditions. Although these conditions can be quantified and checked in practice, our future work aims to relax these conditions, or to learn a communication policy satisfying these conditions. AME utilizes the information redundancy of communication, which can be achieved by the learning of a communication policy in many tasks, as an extension of the current method.

5.7 Supplemental Materials: Proofs and Additional Details

5.7.1 Technical Proofs

For the simplicity of the proof, we make the following definition.

Definition 48. *(Purely Benign k-sample and contaminated k-sample) A k-sample* $[\mathbf{m}]_k \in \mathcal{H}_k(\mathbf{m})$ *is purely benign if every message in* $[\mathbf{m}]_k$ *comes from a benign agent and is unperturbed. A k-sample* $[\mathbf{m}]_k \in \mathcal{H}_k(\mathbf{m})$ *is contaminated if there exists some message in* $[\mathbf{m}]_k$ *that is perturbed.*

For notation simplicity, let $n_1 := |\mathcal{H}_k(\mathbf{m})| = \binom{N-1}{k}$ be the total number of k-samples from a message set $\mathbf{m}$. Note that the total number of purely benign k-samples is $n_2 := \binom{N-C-1}{k}$, and

the total number of contaminated k-samples is $n_1 - n_2 = \binom{N-1}{k} - \binom{N-C-1}{k}$.

5.7.2 Proofs in Discrete Action Space

Action Certificates We first prove the action certificates in the discrete action. For notation simplicity, we slightly abuse notation and use $u_{\max}$ to denote $u_{\max}(\mathbf{m}_{\text{adv}})$. Note that Theorem 41 is a special case of Theorem 46 ($u_{\max} > n_1 - n_2$ and $D = \binom{N-1}{k}$), so we first prove the general version Theorem 46 and then Theorem 41 holds as a result.

Proof of Theorem 46 and Theorem 41. The majority voted action $\tilde{a}$ is a benign action, i.e., $\tilde{a} \in \mathcal{A}_{\text{benign}}$, if the ablation policy $\hat{\pi}$ renders action $\tilde{a}$ for at least one purely benign k-sample. If $u_{\max} > n_1 - n_2$, since $n_1 - n_2$ is exactly the total number of contaminated k-samples, then it is guaranteed that there is at least one purely benign k-sample for which $\hat{\pi}$ renders $\tilde{a}$. Thus, $\tilde{a} \in \mathcal{A}_{\text{benign}}$, and Theorem 41 holds.

On the other hand, if $u_{\max} \leq n_1 - n_2$, then in order for $\tilde{a}$ to be in $\mathcal{A}_{\text{benign}}$, among the $u_{\max}$ k-samples resulting in $\tilde{a}$ there can be at most $u_{\max} - 1$ contaminated k-samples. There are $\sum_{j=0}^{u_{\max}-1} \binom{n_1-n_2}{j}\binom{n_2}{D-j}$ such combinations in terms of the sampling of D, and the total number of combinations are $\binom{n_1}{D}$. Therefore, we get Equation (5.15).

□

Reward Certificate. Next, following Theorem 41, we proceed to prove the reward certificate.

Proof of Corollary 42. Based on the definition of benign action set $\mathcal{A}_{\text{benign}}$, $\widetilde{\pi}$ selects an action $\widetilde{a}$ if and only if there exists a purely benign k-sample $[\mathbf{m}]_k \in \mathcal{H}_k(\mathbf{m})$ such that the message-ablation policy $\hat{\pi}$ selects $\widetilde{a} = \hat{\pi}(\tau, [\mathbf{m}]_k)$. Therefore, for any trajectory generated by $\widetilde{\pi}$ under attacks, there

is a trajectory of $\hat{\pi}$ with a list of k-samples $[\mathbf{m}]_k^{(1)}, [\mathbf{m}]_k^{(2)}, \cdots$ that renders the same cumulative reward under no attack.

□

5.7.3 Proofs in Continuous Action Space

Action Certificate. Similar to the discrete-action case, we first prove Theorem 47, and then prove Theorem 43 as a special case of Theorem 47.

Proof of Theorem 47. To understand the intuition of element-wise median operation in continuous action space, let us first start with an intuitive example: consider 5 arbitrary numbers x_1, ..., x_5, if we already know 3 of them x_1, x_2, x_3, then it is certain that $\min(x_1, x_2, x_3) \leq \mathsf{Median}(x_1, \cdots, x_5) \leq \max(x_1, x_2, x_3)$. Therefore, when purely benign k-samples form the majority (Condition 2), the element-wise median action falls into the range of actions produced by safe messages.

To be more general, in a continuous action space, $\tilde{a} \in \mathsf{Range}(\mathcal{A}_{\text{benign}})$ is equivalent to the condition that out of the D sampled k-samples, purely benign k-samples make up the majority. There are $\sum_{j=\tilde{D}}^{D} \binom{n_2}{j}\binom{n1-n2}{D-j}$ such combinations in terms of the sampling of D, where $\tilde{D} = \lfloor \frac{D}{2} \rfloor + 1$. Once again the total number of combinations is $\binom{n_1}{D}$. Therefore, we get Equation (5.16). □

Proof of Theorem 43. The proof of Theorem 43 follows as a special case of Theorem 47 when $D = \binom{N-1}{k} = n_1$. In this case, the only non-zero term left in the numerator of p_D is $\binom{n_2}{j}\binom{n_1-n_2}{n_1-j} = \binom{n_2}{n_2}\binom{n_1-n_2}{n_1-n_2} = 1$ (we need $n_2 \geq j$ and $n_1 - n_2 \geq n_1 - j$ to keep the numerator from vanishing, which implies $j = n_2$, which is no lower than $\tilde{D}$ since $n_2 > n_1/2$ due to Condition 2). Hence we have $p_D = 1$. □

Reward Certificate. Next, we derive the reward guarantee for the continuous-action case.

Proof of Theorem 45. We let $\mathbb{P}(a|s;\pi)$ be the probability of the message-ablation policy $\hat{\pi}$ taking action a at state s, where π can be either the message-ablation policy $\hat{\pi}$ or the message-ensemble policy $\widetilde{\pi}$. Note that this is a conditional probability function, and the policy does not necessarily observe s.

Without loss of generality, let ν^* be the optimal attacking algorithm that minimizes $\tilde{V}_{\nu}^{\widetilde{\pi}}$. Let $\mathcal{A}_s$ denote the range of benign action at state s induced by the current message-ablation policy $\hat{\pi}$. Then we have

$$
\begin{aligned}
&\sup_{s\in\mathcal{S}}\left|V^{\hat{\pi}}(s)-\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s)\right| \\
&=\sup_{s\in\mathcal{S}}\left|\mathbb{E}_{a\sim\mathbb{P}(a|s;\hat{\pi})}\left[R(s,a)+\gamma\int P(s'|s,a)V^{\hat{\pi}}(s')\mathrm{d}s'\right]-\mathbb{E}_{a\sim\mathbb{P}(a|s;\widetilde{\pi})}\left[R(s,a)+\gamma\int P(s'|s,a)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'\right]\right| \\
&\le\sup_{s\in\mathcal{S}}\sup_{a_1,a_2\in\mathcal{A}_s}\left|R(s,a_1)+\gamma\int P(s'|s,a_1)V^{\hat{\pi}}(s')\mathrm{d}s'-R(s,a_2)-\gamma\int P(s'|s,a_2)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s)(s')\mathrm{d}s'\right| \\
&\le\sup_{s\in\mathcal{S}}\sup_{a_1,a_2\in\mathcal{A}_s}|R(s,a_1)-R(s,a_2)|+\sup_{a_1,a_2\in\mathcal{A}_s}\left|\gamma\int P(s'|s,a_1)V^{\hat{\pi}}(s')\mathrm{d}s'-\gamma\int P(s'|s,a_2)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'\right| \\
&\le\epsilon_R+\gamma\sup_{s\in\mathcal{S}}\sup_{a_1,a_2\in\mathcal{A}_s}\left|\int P(s'|s,a_1)V^{\hat{\pi}}(s')\mathrm{d}s'-\int P(s'|s,a_2)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'\right| \\
&\le\epsilon_R+\gamma\sup_{s\in\mathcal{S}}\sup_{a_1,a_2\in\mathcal{A}_s}\left|\int P(s'|s,a_1)V^{\hat{\pi}}(s')\mathrm{d}s'-\int P(s'|s,a_1)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'\right| \\
&\qquad+\gamma\sup_{s\in\mathcal{S}}\sup_{a_1,a_2\in\mathcal{A}_s}\left|\int P(s'|s,a_1)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'-\int P(s'|s,a_2)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'\right| \\
&\le\epsilon_R+\gamma\sup_{s\in\mathcal{S}}\left|V^{\hat{\pi}}(s)-\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s)\right|+\gamma\left|\int P(s'|s,a_1)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'-\int P(s'|s,a_2)\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s')\mathrm{d}s'\right| \\
&\le\epsilon_R+\gamma\sup_{s\in\mathcal{S}}\left|V^{\hat{\pi}}(s)-\tilde{V}_{\nu^*}^{\widetilde{\pi}}(s)\right|+\gamma V_{\max}\epsilon_P.
\end{aligned}
\tag{5.17}
$$

By solving for the recurrence relation over $\sup_{s\in\mathcal{S}} \left|V^{\hat{\pi}}(s) - \tilde{V}^{\tilde{\pi}}_{\nu^*}(s)\right|$, we obtain

$$\sup_{s\in\mathcal{S}} \left|V^{\hat{\pi}}(s) - \tilde{V}^{\tilde{\pi}}_{\nu^*}(s)\right| \leq \frac{\epsilon_R + \gamma V_{\max}\epsilon_P}{1-\gamma}. \tag{5.18}$$

which leads to the desired relation in Theorem 45.

□

5.7.4 Additional Implementation Details and Empirical Results

5.7.4.1 Implementation Details

In our experiments, we use the Proximal Policy Optimization (PPO) [16] algorithm to train all agents (with parameter sharing among agents) as well as the attackers. Specifically, we adapt from the elegant OpenAI Spinning Up [133] Implementation for PPO training algorithm. On top of the Spinning Up PPO implementation, we also keep track of the running average and standard deviation of the observation and normalize the observation. All experiments are conducted on NVIDIA GeForce RTX 2080 Ti GPUs.

For the policy network, we use a multi-layer perceptron (MLP) with two hidden layers of size 64. For a discrete action space, a categorical output distribution is used. For a continuous action space, since the valid action is bounded within a small range [-0.01,0.01], we parameterize the policy as a Beta distribution, which has been proposed in previous works to better solve reinforcement learning problems with bounded actions [134]. In particular, we parameterize the Beta distribution by α_θ and β_θ, such that $\alpha = \log(1 + e^{\alpha_\theta(s)}) + 1$ and $\beta = \log(1 + e^{\beta_\theta(s)}) + 1$ (1 is added to make sure that $\alpha, \beta \geq 1$). Then, $\pi(a|s) = f(\frac{a-h}{2h}; \alpha, \beta)$, where $h = 0.01$, and

$f(x; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$ is the density function of the Beta Distribution. For the value network, we also use an MLP with two hidden layers of size 64.

In terms of other hyperparameters used in the experiments, we use a learning rate of 0.0003 for the policy network, and a learning rate of 0.001 is used for the value network. We use the Adam optimizer with $\beta_1 = 0.99$ and $\beta_2 = 0.999$. For every training epoch, the PPO agent interacts with the environment for 4000 steps, and it is trained for 500 epochs in our experiments.

Implementation of Attackers An attacker maps its own observation to the malicious communication messages that it will send to the victim agent. Thus, the action space of the attacker is the communication space of a benign agent, which is bounded between -1 and 1.

- *Heuristic Attacker* We implement a fast and naive attacking method for the adversary. At every dimension, the naive attacker randomly picks 1 or -1 as its action, and then sends the perturbed message which consists entirely of 1 or -1 to the victim agent.

- *Adaptive RL Attacker* We use the PPO algorithm to train the attacker, where we set the reward of the attacker to be the negative reward of the victim. The attacker uses a Gaussian policy, where the action is clipped to be in the valid communication range. The network architecture and all other hyperparameter settings follow the exact same from the clean agent training.

Implementation of Baselines

- *Vanilla Learning* For Vanilla method, we train a shared policy network to map observations and communication to actions.

- *Adversarial Training (AT)* For adversarial training, we alternate between the training of attacker and the training of the victim agent. Both the victim and attackers are trained by

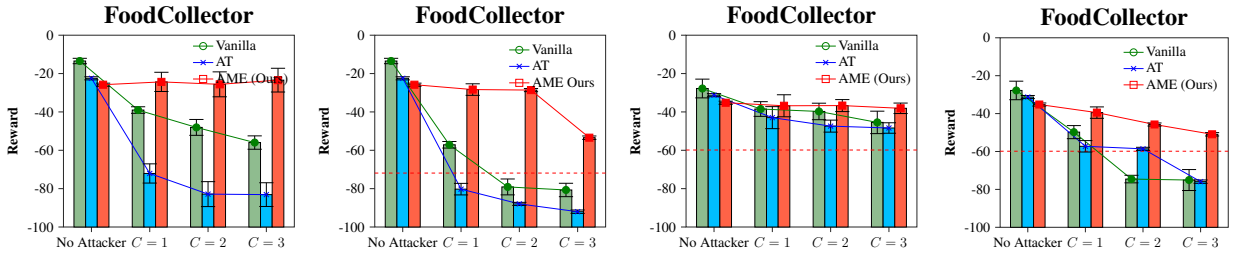
PPO. For every 200 training epochs, we switch the training, where we either fix the trained victim and train the attacker for the victim or fix the trained attacker and train the victim under attack. We continue this process for 10 iterations.

Note that the messages are symmetric (of the same format), we shuffle the messages before feeding them into the policy network for both Vanilla and AT, to reduce the bias caused by agent order. We find that shuffling the messages helps the agent converge much faster (50% fewer total steps). Note that AME randomly selects k-samples and thus messages are also shuffled.

5.7.4.2 Additional Experiment Results

A. Full Results of FoodCollector

Figure 5.4 below shows the results of both discrete-action FoodCollector and continuous-action FoodCollector.



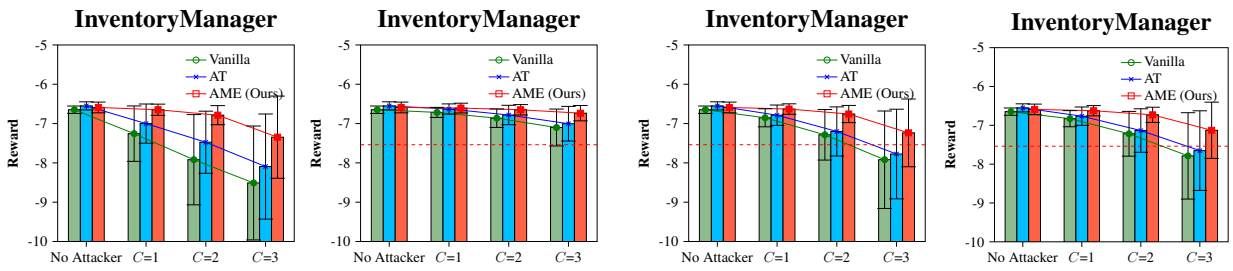
(a) Disc. & Heuristic (b) Disc. & Adaptive (c) Cont. & Heuristic (d) Cont. & Adaptive

Figure 5.4: Rewards of our AME and baselines in FoodCollector, under no attacker and varying numbers of adversaries for adaptive and heuristic (random message) attacks.

B. Full Results of InventoryManager

Figure 5.5 below shows the results of InventoryManager under adaptive attacks and several heuristic attacks.

C. Full Results of MARL-MNIST



(a) Adaptive Attack (b) Perm-Attack (c) Swap-Attack (d) Flip-Attack

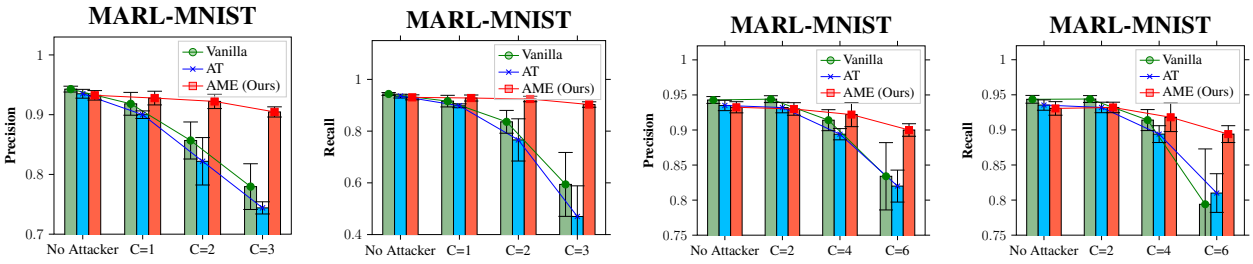
Figure 5.5: Rewards of our AME and baselines in InventoryManager, under no attacker and varying numbers of adversaries for adaptive and various heuristic attacks.

Figure 5.6 demonstrates the robust performance of the MARL algorithm proposed by [128] with our AME defense or baseline defenses (Vanilla and AT). We set $N = 9$ and $k = 2$ for all experiments. Under learned adaptive attackers, the original MARL classifier (Vanilla) [128] without AME suffers from significant performance drop in terms of both precision and recall. Defending with adversarial training (AT) does not achieve good robustness, either. But AME considerably improves the robustness of agents across different numbers of attackers.

Under random attacks, we find that the original MARL classifier [128] is moderately robust when a few communication signals are randomly perturbed. However, when noise exists in many communication channels (e.g. $C = 6$), the performance decreases a lot. In contrast, our AME still achieves high performance when $C = 6$ communication messages are corrupted, even if the guarantee of ablation size k only holds for $C \leq 2$. Therefore, we again emphasize the the theoretical guarantee considers the worst-case strong attack, while under a relatively weak attack, we can achieve better robustness beyond what the theory suggests.

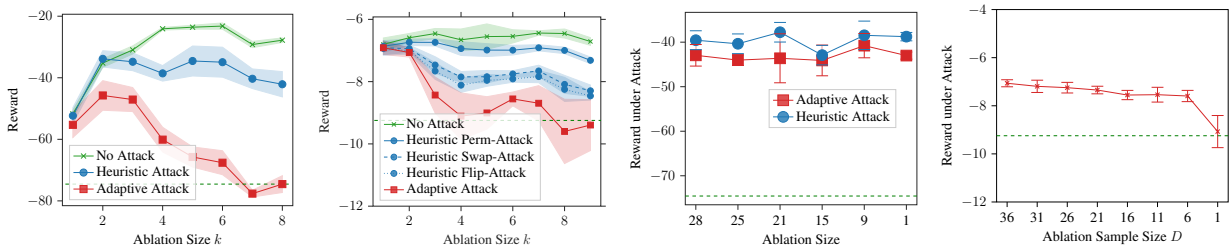
D. Additional Results of Hyperparameter Test

In addition to Figure 5.3, we also provide the plot for hyper-parameter tests in discrete-action FoodCollector and InventoryManager in Figure 5.7 below.



(a) Adaptive Attack (b) Adaptive Attack (c) Heuristic Attack (d) Heuristic Attack

Figure 5.6: **(MARL-MNIST)**: Precision and recall of MARL classification on MNIST without or with AME, under learned adaptive attacks and non-adaptive random attacks. All results are averaged over 5 random seeds.



(a) FoodCollector (b) InventoryManager (c) FoodCollector (d) InventoryManager

Figure 5.7: Hyper-parameter tests of ablation size k and sample size D. We show how natural reward and attacked performance change with **(a)** various k in continuous-action FoodCollector, **(b)** various D in continuous-action FoodCollector, **(c)** various k in InventoryManager, and **(d)** various D in InventoryManager.
Dashed green lines refer to the performance of Vanilla agent under $C = 2$ attacks.

Chapter 6: Adapting to Novel Environments with Theoretical Guarantees

6.1 Introduction

Multi-task reinforcement learning (MTRL) [135, 136, 137] requires the agent to efficiently tackle a series of tasks. A key goal of MTRL is to improve per-task learning efficiency compared against single-task learners, by using the knowledge obtained from previous tasks to learn new tasks. Despite the recent rapid progress in MTRL, some issues remain unsettled. *(1) Guaranteed sample efficiency.* Only a few existing methods have guarantees on sample efficiency, the most common bottleneck of RL algorithms. *(2) Correctness v.s. efficiency.* An overly aggressive application of previous knowledge may transfer incorrect knowledge and deteriorate the performance on new tasks, resulting in a "negative transfer" [138]. However, if an agent is overly conservative in applying previously learned knowledge, much of the similarities between tasks will be ignored, resulting in an "inefficient transfer". It is nontrivial to balance between the correctness and efficiency or achieve both. *(3) Varying state/action space across tasks.* In practice, transferring knowledge learned from smaller environments to learning in larger environments is extremely useful. However, most existing works on MTRL assume the state/action space is shared across tasks.

In an effort to provide guaranteed sample efficiency for MTRL, Brunskill et al. [136] propose an algorithm that clusters the underlying Markov Decision Processes (MDPs) of tasks into

groups and identifies new tasks as learned groups. However, transferring knowledge from the clustered MDP models could be an "inefficient transfer" if the underlying models are too different to be clustered into a small number of groups. Similarly, most existing model-based approaches [137, 139] only exploit model-level similarities, which also makes it difficult to transfer knowledge among different-sized tasks.

We remedy the aforementioned three issues by extraction of more commonalities in tasks without suffering from "negative transfer". A motivating example is the navigation problem in mazes with slippery floors which result in stochastic transitions. For instance, the agent taking an action of going *up* on ice could slip to the *left*, *right* or *down* (instead of *up*) with a certain probability determined by the slipperiness of ice. The slipperiness of the floor depends on the landform of the location, such as sand, marble and ice. We show some examples of different combinations/distributions of the landforms in the maze in Figure 6.1; the MDP models are drastically different across different mazes, therefore transferring knowledge using similarity of models is inefficient.

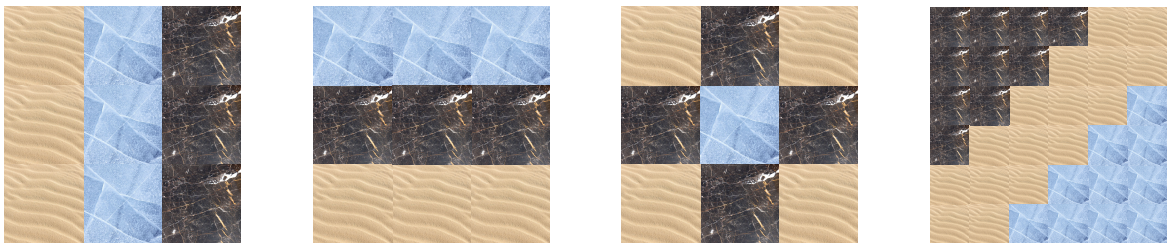


Figure 6.1: Examples of landform combinations in Maze, where ▒ stands for sand, ▓ stands for marble and ▒ stands for ice. Different landforms have different slippery probability, thus different transition dynamics. Consider a $\sqrt{S} \times \sqrt{S}$ maze with G types of landforms. There could be up to G^S different MDP models, making it prohibitive to extract similarities from the models. However, the types of underlying transition dynamics associated with each state/location are governed by the number of distinct landforms G.

However, our key observation is that the same landforms share the transition dynamics, and knowledge could be transferred from sand to sand, marble to marble, and ice to ice. More

importantly, we can extend the knowledge learned from a maze to any-sized mazes consisting of these same types of landforms (e.g., the 4th example in Figure 6.1). With this idea, we achieve more effective and efficient knowledge transfer by exploiting similarities at the level of *state-action transition dynamics* **instead of** *MDP model dynamics*, allowing knowledge transfer between tasks with **varying state/action space without prior knowledge of inter-task mappings.** The challenge of learning is now reduced to extracting such "landforms" without prior knowledge of the tasks.

We propose a novel method called *Template Learning (TempLe)* for MTRL, which provably guarantees sample efficiency and achieves efficient transfer learning for multi-task reinforcement learning with varying state/action space. We extract templates for similar state-action transition dynamics (landforms in the example above), called *Transition Templates*, and confidently improve the efficiency of transition dynamics estimation in new tasks. By sharing experience among state-action pairs associated with similar templates, the learning process is expedited. We introduce two versions of TempLe: one is for online MTRL without prior knowledge about models, named *Online Template Learning (O-TempLe)*, the other further improves the learning efficiency based on a finite-model assumption, named *Finite-Model Template Learning (FM-TempLe)*.

Summary of Contributions.

(1) TempLe achieves a significant *reduction of sample complexity* compared with state-of-the-art PAC-MDP (Probably Approximately Correct in Markov Decision Processes) algorithms.

(2) TempLe covers two realistic settings, solving MTRL problems in different regimes – *with or without prior knowledge of models*.

(3) To the best of our knowledge, TempLe is the *first PAC-MDP algorithm* that is able to learn tasks with *varying state/action spaces* without any prior knowledge of inter-task mappings.

6.2 Related Work

PAC-MDP MTRL Algorithms. Brunskill et al. [136] present the first formal analysis of the sample complexity for MTRL. They propose a two-phase algorithm and prove that per-task sample complexity is reduced compared with single-task learners. However, they require all tasks coming from a small number of models, and when the number of distinct models is large, their algorithm becomes similar to single-task learning. In this chapter, we show our proposed methods outperform the method provided by [136] both in theory and in experiments. There are other PAC-MDP algorithms for multi-task RL, considering the problem from different perspectives. For example, Brunskill et al. [140] discuss lifelong learning in semi-Markov decision processes (SMDPs), where options are involved. Liu et al. [139] extend the finite-model method [136] to continuous state space. Feng et al. [141] and Tirinzoni et al. [142] significantly reduce the sample complexity, but are under the assumption of generative models. Modi et al. [137] improve the learning efficiency through the assistance of side informations. Abel et al. [143] propose MaxQInit, which transfers the maximum Q values across tasks. We empirically compare with MaxQInit in this chapter.

Reducing MDPs to Compact Ones. There is a line of research that reduces the original MDPs to compact ones to achieve sample efficiency, including Relocatable Action Model (RAM) [144], homomorphism [145], and ϵ-equivalent MDP [146]. However, since learning such compact structures is usually difficult (e.g., learning homomorphism is NP-hard as noted by [147]), most of the previous works require some prior knowledge. To give a detailed comparison, **our algorithm (1) requires no prior knowledge about the MDP structure.** RAM [144] requires knowledge of the "type" of all states (walls, pits, etc) and the next-state function of all states and type-action

outcomes. Its continuous extension [148] also needs knowledge of the types. Homomorphism works [145, 147, 149] require knowledge of (candidate) homomorphisms to compress an SMDP or transfer knowledge between MDPs. **(2) works for general RL problems with PAC guarantee.** Although [150] (learns latent structure by clustering) and [151] (learns soft homomorphisms) provide methods that do not require knowledge of the structure, Leffler et al. [150] study a simplified non-MDP problem where actions do not influence state transitions, and Sorg et al. [151] do not provide theoretical guarantees when the target model is not known in advance.

Overall, our method is different from the above works, as we do not pre-define the compact structure. Instead, we observe that the transition dynamics, if permuted into descending order, could be naturally grouped to some template. Notably, we *learn* the similarities rather than assuming knowledge of them. Our method could be more practical than the above works [144, 145, 147, 148, 149, 150, 151] in multi-task RL, since a new task is often drawn randomly and knowing its structure in advance could be unrealistic.

Comparison with C-UCRL [152]. C-UCRL learns a single task by leveraging a state-action equivalence structure that is similar with our proposed templates. They provide an improved regret bound in the case of a known equivalence structure. However, in the more challenging case of an unknown equivalence structure, as is the setting of our chapter, no regret bound is provided. In contrast, our work provides a sample complexity guarantee under the unknown equivalence structure scenario. In addition, C-UCRL does not extend trivially to multi-task setting since it find a coarse partition of all state-action pairs at every step, while in MTRL, new state-action pairs come with new tasks, and negative transfer problem may exist when the equivalence structure is unknown.

6.3 Background and Problem Setup

Sample Complexity. The general goal of RL algorithms is to learn an optimal policy for an MDP with as few interactions as possible. A widely-used framework to evaluate the performance of RL algorithms is *sample complexity of exploration* [153], or *sample complexity* for short. For any $\epsilon > 0$ and any step $h > 0$, if the policy π_h generated by an RL algorithm L satisfies $V^* - V^{\pi_h} \leq \epsilon$, we say L is near-optimal at step h. If for any $0 < \delta < 1$, the total number of steps that L is not near-optimal is upper bounded by a function $\zeta(\epsilon, \delta)$ with probability at least $1 - \delta$, then ζ is called the *sample complexity* [153] of L.

Multi-Task RL. We study a common multi-task RL (MTRL) setting, where an agent interacts with multiple tasks streaming in, each of which is corresponding to a specific MDP. The tasks are i.i.d. drawn from a set $\mathcal{M}$ of MDPs (models). MDPs in $\mathcal{M}$ may have different dynamics or state/action space. We consider two different cases and propose different algorithm variants for them: (1) the number of MDPs in the set $|\mathcal{M}|$ can be very large or infinite; (2) $|\mathcal{M}|$ is finite and not large.

6.4 Proposed Approach

6.4.1 Learning with Templates

As motivated in the example described in Section 6.1, the main idea of this work is to boost the learning process by aggregating similar state-action transition dynamics (see Definition 49). We permute the elements of transition dynamics/probability vectors to be in descending order, and aggregate these permuted transition probabilities to obtain "templates of transition" defined

in Definition 50. We show that the templates are effective abstractions of the environment.

6.4.1.1 Transition Template: An Abstraction of Dynamics

In this section, we introduce a more compact way to represent the model dynamics of an MDP. We first formally define the transition dynamics of a state-action (s-a) pair.

Definition 49 (State-Action (s-a) Transition Dynamics)**.** *For any state-action pair* (s, a)*, its* ***transition dynamics*** *is defined as a length-*$(S+1)$ *vector*

$$\theta(s,a) = [p(s_1|s,a), p(s_2|s,a), \cdots, p(s_S|s,a), r(s,a)]$$

, where S *is the number of states.*

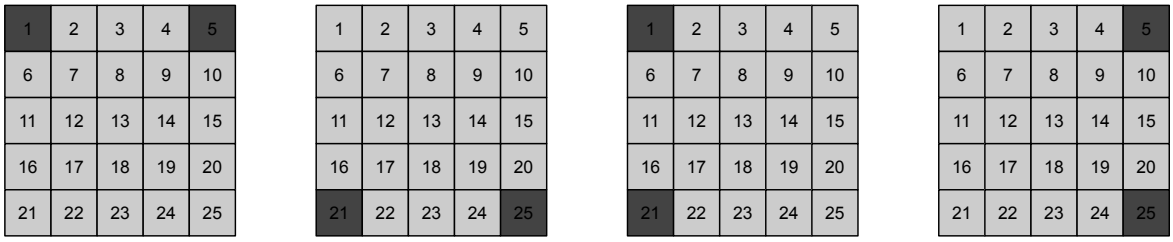
Note that s-a transition dynamics are different from the model dynamics, which characterize the transitions for all s-a pairs. In s-a transition dynamics, the first S elements form the transition probability vector $p(\cdot|s,a)$. As defined in most RL literatures [136, 153], the order of elements in $p(\cdot|s,a)$ is the natural order of the states. In contrast, we re-order the elements of $p(\cdot|s,a)$ by their values, and obtain a more compact representation of the transition dynamics called *Transition Template*.

Definition 50 (Transition Template)**.** *A* ***Transition Template (TT)*** $\mathbf{g}$ *is defined as a tuple* $(\mathbf{g}^{(p)}, g^{(r)})$*, where* $\mathbf{g}^{(p)} \in \mathbb{R}^S$ *is a transition probability vector with non-increasingly ordered elements, i.e.,* $\sum_{i=1}^{S} g_i^{(p)} = 1$ *and* $g_i^{(p)} \geq g_j^{(p)} \geq 0, \forall 1 \leq i \leq j \leq S$*;* $0 \leq g^{(r)} \leq 1$ *is a scalar representing the reward.*

Any s-a transition dynamics can be permuted to an unique TT by re-arranging the tran-

sition probability vector $p(\cdot|s,a)$ in a decreasing order and maintaining the reward $r(s,a)$ to $g^{(r)}$, i.e., $\mathbf{g}_{(s,a)} = (\text{desc}(p(\cdot|s,a)), r(s,a))$, where desc orders the elements of $p(\cdot|s,a)$ from the largest value to the smallest value. For example, if $\theta(s_1,a_1) = [0.3, 0.7, 0, 1]$, and $\theta(s_2,a_2) = [0, 0.3, 0.7, 1]$, then (s_1,a_1) and (s_2,a_2) have the same TT $([0.7, 0.3, 0], 1)$, although their s-a transition dynamics are different.

A TT is a representation of multiple s-a transition dynamics with some similarities. It ignores how the s-a pair transits to a specific next state, but only considers the patterns of transition probabilities, allowing more efficient exploitation of similarities.



(a) action='↑' (b) action='↓' (c) action='←' (d) action='→'

Figure 6.2: An example of TTs in a 5×5 slippery gridworld with no reward and slipping probability=0.4. The template at all ■s is $\mathbf{g}_1 = ([0.8, 0.2, 0, \cdots, 0], 0)$, and the template at all □s is $\mathbf{g}_2 = ([0.6, 0.2, 0.2, 0, \cdots, 0], 0)$.

An Intuitive Example. Consider a 5×5 gridworld where the agent has 4 actions: $\uparrow, \downarrow, \leftarrow$ and $\rightarrow$, as well as 25 states, as shown in Figure 6.2a, 6.2b, 6.2c and 6.2d. Thus there are 100 distinct state-action pairs in total. Since the slipping probability is 0.4, which means action $\uparrow$ will become $\leftarrow$ or $\rightarrow$ with probability 0.2 respectively, we know the transition probability $p(\cdot|s=1, a=\uparrow)$ is

$$p(s'|s=1,a=\uparrow) = \begin{cases} 0.8 & \text{if } s'=1 \\ 0.2 & \text{if } s'=2 \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

By re-ordering, its TT is $([0.8, 0.2, 0, \cdots, 0], 0)$.

Similarly, for state 2 and action $\uparrow$,

$$p(s'|s=2, a=\uparrow) = \begin{cases} 0.6 & \text{if } s'=2 \\ 0.2 & \text{if } s'=1 \text{ or } 3 \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

Its TT is $([0.6, 0.2, 0.2, 0, \cdots, 0], 0)$.

For state 3 and action $\uparrow$,

$$p(s'|s=3, a=\uparrow) = \begin{cases} 0.6 & \text{if } s'=3 \\ 0.2 & \text{if } s'=2 \text{ or } 4 \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

Its TT is also $([0.6, 0.2, 0.2, 0, \cdots, 0], 0)$.

In this way, we can find that there are only 2 distinct TTs, like shown in the figure, which is much less than the number of state-action pairs. The state-action pairs with the same TTs are able to share the "relative" probability of transitioning.

6.4.1.2 Empirical Estimation of Transition Templates

Section 6.4.1.1 defines TT based on the underlying s-a transition dynamics. However, in reality, we do not have access to the underlying dynamics. In model-based RL, a key step is to estimate the dynamics and to build a model of the environment. We now illustrate the estimation

of TTs, as well as how TTs augments the learning process.

The conventional estimation of s-a transition dynamics. A direct estimate of $\theta(s,a)$ is obtained through experience, $\hat{\theta}(s,a) = [\frac{n(s,a,s_1)}{n(s,a)}, \frac{n(s,a,s_2)}{n(s,a)}, \cdots, \frac{n(s,a,s_S)}{n(s,a)}, \frac{R(s,a)}{n(s,a)}]$, where $n(s,a,s')$ is the number of observations of transitioning from s to s' by taking action a, $n(s,a)$ is the total number of observations of (s,a), and $R(s,a)$ is the cumulative rewards obtained by (s,a). An accurate estimate of the transition dynamics $\theta(s,a)$ requires a large enough number of observations $n(s,a)$ according to the theory of concentration bounds. Therefore, it is sample-consuming to accurately estimate the transition dynamics of each s-a pair in this way.

Augmented estimation of s-a transition dynamics. As discussed in Section 6.4.1.1, different s-a pairs may share the same TTs. Our goal is then to aggregate the estimations of s-a transition dynamics associated with the same TTs. We introduce the following process to obtain estimates of all s-a transition dynamics:

(1) rough estimation: obtain $\widehat{\theta}(s,a) = [\frac{\mathbf{n}(s,a,\cdot);R(s,a)}{n(s,a)}]$ for each (s,a) with a small n;

(2) permutation: permute each $\widehat{\theta}(s,a)$ to its corresponding permuted estimates $\tilde{\boldsymbol{g}}_{(s,a)}$;

(3) template identification: identify the group of the permuted estimate $\tilde{\boldsymbol{g}}_{(s,a)}$ such that permuted estimates are similar within the group, and obtain a more confident estimate of TT $\widehat{\mathbf{g}}$ aggregating within-group statistics.

(4) augmentation: for every (s,a), obtain a more confident estimate of the transition dynamics by permuting back its corresponding TT with accumulated knowledge.

The noisy estimate of transition dynamics will not render error other than the smaller amount of noise in estimated transition templates if it is identified into the right group. To guarantee accurate identification, the ordering of the elements in the noisy estimate should be consistent with the ground truth. Therefore, the consistency of our estimation depends on TT gap as defined

in Definition 51 and "ranking gap" as defined in Definition 53 in Section 6.4.4.

Now we are ready to formally introduce our algorithms in two settings, Online MTRL and Finite-Model MTRL.

6.4.2 O-TempLe: Online Template Learning

In the *online MTRL setting*, an agent interacts with multiple tasks streaming-in, each of which corresponding to a specific MDP. The tasks are i.i.d. drawn from a set $\mathcal{M}$ of MDPs (models). MDPs in $\mathcal{M}$ may have different state/action spaces. The number of MDPs $|\mathcal{M}|$ can be arbitrarily large.

We introduce Online Template Learning (O-TempLe) for the online MTRL setting. O-TempLe is a meta-learning algorithm with model-based "*base learners*" which compute policies for the current task. We use RMax [154] as the base learner, and it can be replaced by other model-based methods such as E^3 [155] and MBIE [156]. The principle of RMax algorithm on an MDP M is to build an induced MDP based on a known threshold m. A state-action pair is said to be m-known if the number of visits/observations $n(s,a) \geq m$. A state is m-known if $n(s,a) \geq m, \forall a \in \mathcal{A}$. The set of all m-known states induces an MDP M_k, where for any m-known state s, $p(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$, $r(s,a) = \frac{R(s,a)}{n(s,a)}$ and for any non-m-known state s, $p(s'|s,a) = \mathbb{I}\{s' = s\}$, $r(s,a) = R_{\max}$. Then, RMax computes an optimal policy based on the optimistic model by dynamic programming.

In contrast, O-TempLe uses augmented estimation introduced in Section 6.4.1.2. to reduce the required number of visits to every single s-a pair. Instead of aggregating the estimates of all s-a transition dynamics at once, O-TempLe asynchronously identifies the TTs of s-a pairs and

updates the template groups in an online manner, through measuring the distances among TTs.

Algorithm 11: Online Template Learning (O-TempLe)

1 **Input:** user-specified TT gap $\hat{\tau}$; error tolerance ϵ; discount factor γ; regular known threshold m; small known threshold m_s
2 Initialize an empty TT group set $\mathcal{G}$ and TT visit set $\mathcal{O}$
3 **for** $t \leftarrow 1, 2, \cdots$ **do**
4 Receive a task M_t
5 Initialize visits $\boldsymbol{n}(s, a, \cdot) \leftarrow \mathbf{0}$, accumulative rewards $R(s, a) \leftarrow 0$, $\forall (s, a) \in (\mathcal{S}, \mathcal{A})$, an empty known state-action set $\mathcal{K}$, and an initial policy π
6 **for** $h \leftarrow 1, 2, \cdots, H$ **do**
7 Take action $a_h \leftarrow \pi(s_h)$, get s_{h+1} and r_h
8 Update visits $n(s_h, a_h, s_{h+1})$ and $R(s_h, a_h)$
  `// TT identification with the small threshold`
9 **if** $(s_h, a_h) \notin \mathcal{K}$ ***and*** $\|\boldsymbol{n}(s_h, a_h, \cdot)\|_{\ell_1} = m_s$ **then**
10 $\tilde{\mathbf{g}}, \mathbf{o}_{\tilde{\mathbf{g}}}, \sigma \leftarrow$ `gen-TT` $(\boldsymbol{n}(s_h, a_h, \cdot), R(s_h, a_h))$
11 **if** *no* $\mathbf{g} \in \mathcal{G}$ *is* $\hat{\tau}$*-close to* $\tilde{\mathbf{g}}$ **then**
12 Add $\tilde{\mathbf{g}}$ to $\mathcal{G}$, $\mathbf{o}_{\tilde{\mathbf{g}}}$ to $\mathcal{O}$
13 **else**
14 Find the closest TT $\mathbf{g}^*$ to $\tilde{\mathbf{g}}$
15 `TT-update` $(\mathbf{g}^*, \mathbf{o}_{\mathbf{g}^*}, \boldsymbol{n}(s_h, a_h, \cdot), R(s_h, a_h))$
16 `augment` $(\mathbf{o}^*_{\mathbf{g}}, \boldsymbol{n}(s_h, a_h, \cdot), R(s_h, a_h), \sigma)$
  `// policy update with the regular threshold`
17 **if** $(s_h, a_h) \notin \mathcal{K}$ ***and*** $\|\boldsymbol{n}(s_h, a_h, \cdot)\|_{\ell_1} \geq m$ **then**
18 Update π using visits $\boldsymbol{n}$ and R by RMax
19 Add (s_h, a_h) to $\mathcal{K}$
20 **for** *all* $(s, a) \in (\mathcal{S}, \mathcal{A})$ *with identified* TT $\mathbf{g}_{(s,a)}$ **do**
21 `TT-update` $(\mathbf{g}_{(s,a)}, \mathbf{o}_{\mathbf{g}_{(s,a)}}, \boldsymbol{n}(s, a, \cdot), R(s, a))$
22 **Output:** Near-optimal policies $\{\pi_t\}_{t=1,2,\cdots}$

Algorithm 11 illustrates how O-TempLe works. In addition to the regular known threshold m used in RMax, we design a smaller known threshold m_s, which is the smallest number of visits to ensure identifying the TTs of all s-a pairs. If for any (s, a), the total number of visits ($\|\boldsymbol{n}(s, a, \cdot)\|_{\ell_1}$) reaches m_s, then the estimated TT $\tilde{\mathbf{g}}$ of (s, a) will be generated by function GEN-TT. If $\tilde{\mathbf{g}}$ has at least $\hat{\tau}$-distance with all existing TTs, we regard it as a new TT and append it to set $\mathcal{G}$ (Line 10-11); otherwise (Line 12-15), we find the closest TT to $\tilde{\mathbf{g}}$, then synchronize the experience of (s, a) in the current task and the accumulated experience that its TT holds by

Algorithm 12: TT Functions

1 **Function** `gen-TT` $(\boldsymbol{n}, R)$ **:**
  `// generate TT`
2 Find permutation σ s.t. $\sigma(\boldsymbol{n})$ is in descending order
3 Ordered visits $\mathbf{o}_\mathbf{g}^{(N)} \leftarrow \sigma(\boldsymbol{n})$, $o_\mathbf{g}^{(R)} \leftarrow R$, $\mathbf{o}_\mathbf{g} \leftarrow (\mathbf{o}_\mathbf{g}^{(N)}, o_\mathbf{g}^{(R)})$
4 transition template $\mathbf{g} \leftarrow (\frac{\mathbf{o}_\mathbf{g}^{(N)}}{\|\boldsymbol{n}\|_{\ell_1}}, \frac{o_\mathbf{g}^{(R)}}{\|\boldsymbol{n}\|_{\ell_1}})$
5 **return** $\mathbf{g}, \mathbf{o}_\mathbf{g}, \sigma$
6 **Function** TT`-update` $(\mathbf{g}, \mathbf{o}_\mathbf{g}, \boldsymbol{n}, R)$ **:**
  `// add visits to TT`
7 $\mathbf{o}_\mathbf{g} \leftarrow \mathbf{o}_\mathbf{g} + (descending(\boldsymbol{n}), R)$
8 $\mathbf{g} \leftarrow (\frac{\mathbf{o}_\mathbf{g}^{(N)}}{\|\mathbf{o}_\mathbf{g}^{(N)}\|_{\ell_1}}, \frac{o_\mathbf{g}^{(R)}}{\|\mathbf{o}_\mathbf{g}^{(N)}\|_{\ell_1}})$
9 **Function** `augment` $(\mathbf{o}_\mathbf{g}, \boldsymbol{n}, R, \sigma)$ **:**
  `// augment visits by TT`
10 Get the permutation σ that orders $\boldsymbol{n}$ descendingly
11 $\boldsymbol{n} \leftarrow \boldsymbol{n} + \sigma^{-1}(\mathbf{o}_\mathbf{g}^{(N)})$, $n \leftarrow n + \|\boldsymbol{n}\|_{\ell_1}$
12 $R \leftarrow R + o_\mathbf{g}^{(R)}$

calling functions TT-UPDATE and AUGMENT, which respectively send the current visits of (s, a) to the corresponding TT, and feed the accumulative visits of the TT to the current (s, a). GET-TT, TT-UPDATE and AUGMENT involve the permutation operations, and are given by Algorithm 12. Accumulated experience of each TT is stored in a tuple $\mathbf{o}_\mathbf{g} = (\mathbf{o}_\mathbf{g}^{(N)}, o_\mathbf{g}^{(R)})$, where $\mathbf{o}_\mathbf{g}^{(N)}$ is the total visits accumulated by permuted $\mathbf{n}(s, a, s')$ of all (s, a)'s with TT $\mathbf{g}$. When (s, a) is m-known, the policy is updated (Line 16-18). Overall, our O-TempLe allows grouped s-a transition dynamics to share their visit counts, making it much easier for them to reach m visits than in regular RMax.

Note that Algorithm 11 also works for tasks with varying state/action space, since the comparison of TTs considers the non-zero elements of the transition vectors only. One can compute the difference between two different-sized TTs by simply padding zeros to the end of the shorter TT.

6.4.3 FM-TempLe: Finite-Model Template Learning

Online MTRL setting requires no prior knowledge of the types of underlying MDPs and improves the sample efficiency by accumulating knowledge with TT groups. However, under a more restrictive assumption that the number of possible MDPs $C = |\mathcal{M}|$ is known and small, it is possible to get rid of the dependence on the size of state-action space and achieve *more efficient learning*.

We propose Finite-Model Template Learning (FM-TempLe), an extension of our O-TempLe, under the *finite-model MTRL setting*, where the agent still interacts with streaming-in tasks drawn from a set $\mathcal{M}$ of MDPs, but the number of MDPs in the set $\mathcal{M}$ is small and known.

In contrast with O-TempLe, FM-TempLe is able to correctly identify the TTs of some s-a pairs before they are visited for m_s times. This is because the number of underlying models is small, and thus identifying the model is easy and inexpensive. It is possible to obtain the TTs for all s-a pairs immediately after identifying the model, since the way how TTs are distributed over all s-a pairs is fixed for each MDP model.

The main steps of FM-TempLe are stated below. *(1) Collecting Models:* for the first T_1 tasks, the agent acts in the same way as O-TempLe, but also stores the TT structure of each model. *(2) Grouping Models:* the first T_1 tasks are clustered into finite groups of models based on their TT structures. *(3) Identifying Models:* for any new task, the agent still follows O-TempLe, but also seeks the true model for the current task from all the model groups, by ruling out the groups of models that have different TT structures.

Brunskill et al. [136] make the same finite-model assumption and propose an algorithm FMRL which extracts model similarities. However, FMRL can not transfer knowledge between

two models which are the same except for one state-action pair. In contrast, our FM-TempLe extracts state-action dynamics similarities and thus transferring happens among any state-action pairs that have similar dynamics. Compared with FMRL, FM-TempLe not only has lower sample complexity as proved in Section 6.4.4, but also saves computations due to the direct comparison of TTs.

6.4.4 Theoretical Analysis

This section provides sample complexity analysis of the proposed two algorithms O-TempLe and FM-TempLe. Although O-TempLe and FM-TempLe can be applied to tasks with varying state/action spaces, we assume all tasks have the same $\mathcal{S}$ and $\mathcal{A}$ for simplicity of notations, and the analysis extends to varying state/action spaces trivially.

We first assume there is a diameter D such that any state s' is reachable from any states s in at most D steps on average. This assumption is commonly used in RL [157], and it ensures the reachability of all state from any state on average.

We further define the underlying minimal ℓ_2-distance among TTs as τ, namely TT gap. We also define ν as the ranking gap; a large ranking gap implies that for any s-a pair, the probabilities of transitioning to any two states are substantially different. For any $\mathbf{g} \in \mathcal{G}$, if $\mathbf{g}_i^{(p)} > \mathbf{g}_j^{(p)}$ are two adjacent elements in $\mathbf{g}^{(p)}$, then either $\mathbf{g}_i^{(p)} - \mathbf{g}_j^{(p)} \geq \nu$, or $\mathbf{g}_i^{(p)} - \mathbf{g}_j^{(p)} \leq \tilde{\mathcal{O}}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$(logarithmic terms are hided in $\tilde{\mathcal{O}}(\cdot)$). The ranking gap implies that for any s-a pair, the probabilities of transitioning to any two states are either very close, or substantially different. Note that the algorithms take a user-specified τ, but do not require input of ν.

Definition 51 (TT Gap)**.** *Define the TT distance between two TTs* $\mathbf{g}_a$ *and* $\mathbf{g}_b$ *as* $\rho(\mathbf{g}_a, \mathbf{g}_b) =$

$\|\mathbf{g}_a^{(p)} - \mathbf{g}_b^{(p)}\|_2 + |g_a^{(r)} - g_b^{(r)}|$. *Suppose there is a minimum TT distance* τ*, such that for any two different TTs* $\mathbf{g}_a, \mathbf{g}_b \in \mathcal{G}$*,* $\rho(\mathbf{g}_a, \mathbf{g}_b) \geq \tau$*. We name* τ *as TT gap.*

Remark. According to Lemma 56 in Section 6.7.1.1, the TT gap between any two TTs will not exceed 2 (suppose reward is in between 0 and 1).

The permutation from an s-a transition dynamics to a TT is recorded by a ranking permutation defined as below.

Definition 52 (Ranking Permutation)**.** *For an s-a pair* (s, a) *with transition probability vector* $\mathbf{p} \in \mathbb{R}^S$ *where* $p_i = p(s_i|s, a)$*, by sorting its elements from the largest to the smallest value, we get an ordered vector* $\mathbf{g}^{(p)}$*. Define function* $\sigma : \{1, \cdots, S\} \to \{1, \cdots, S\}$ *as a mapping from ranking to the indices in* $\mathbf{p}$*. For example,* $\sigma(i)$ *is the index of the* i*-th largest element of* $\mathbf{p}$*, i.e.,* $\mathbf{g}_i^{(p)} = \mathbf{p}_{\sigma(i)}$*. The inverse function* σ^{-1} *maps indices to ranking. So* $\sigma^{-1}(j)$ *is the ranking of* $\mathbf{p}_j$*, i.e.,* $\mathbf{g}_{\sigma^{-1}(j)}^{(p)} = \mathbf{p}_j$*. The way way? ordering is unique if for any* $\mathbf{p}_i = \mathbf{p}_j$ *and* $i < j$*, we put* $\mathbf{p}_i$ *before* $\mathbf{p}_j$ *in* $\mathbf{g}^{(p)}$*. As a result,* $\sigma(\cdot)$ *is a bijection and can be regarded as a permutation. We call it* ***ranking permutation*** *of* (s, a)*.*

For simplicity, we slightly abuse notation and use $\sigma(\mathbf{p})$ to denote the re-ordered vector. Thus $\mathbf{g}^{(p)} = \sigma(\mathbf{p})$ and $\mathbf{p} = \sigma^{-1}(\mathbf{g}^{(p)})$.

Definition 53 (Ranking Gap)**.** *Define* ν *as the minimal notable ranking gap, such that for any* $\mathbf{g} \in \mathcal{G}$*, if* $\mathbf{g}_i^{(p)} > \mathbf{g}_j^{(p)}$ *are two adjacent elements in* $\mathbf{g}^{(p)}$ *and* $\mathbf{g}_i^{(p)} - \mathbf{g}_j^{(p)} \geq \mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$*, then* $\mathbf{g}_i^{(p)} - \mathbf{g}_j^{(p)} \geq \nu$ *holds. In other words, two adjacent elements of* $\mathbf{g}^{(p)}$ *satisfy either* $\mathbf{g}_i^{(p)} - \mathbf{g}_j^{(p)} \geq \nu$ *or* $\mathbf{g}_i^{(p)} - \mathbf{g}_j^{(p)} \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$.

If two adjacent elements are different by no more than $\mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$, then the corruption can

be ignored because it will not influence the value of the policy too much, as proved in Lemma 61. Otherwise we need adequate samples to make sure the ranking of elements will succeed.

For notation simplicity, let ω denote $\max\{\min(\tau, \nu), \mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})\}$. Then, we present the sample complexity results of O-TempLe.

Theorem 54 (Sample Complexity of O-TempLe). *For any given $\epsilon > 0$, $1 > \delta > 0$, running Algorithm 11 on T tasks, each for at least $\mathcal{O}(\frac{DSA}{\omega^2} \ln \frac{1}{\delta})$ steps, generates at most $\tilde{\mathcal{O}}\left(\frac{SGV_{\max}^3}{\epsilon^3(1-\gamma)^3} + \frac{TSAV_{\max}}{\omega^2\epsilon(1-\gamma)}\right)$ non-ϵ-optimal steps, with probability at least $1-\delta$, where G is the total number of TTs.*

Remark. (1) Our provided bound achieves *state-of-the-art* dependence on the environment size T, S, A for general MTRL, given that G is independent of T, S, A. (2) When ϵ is small, the sample complexity only has a linear dependence on the number of states S and the number of templates G, because the first term dominates. By definition, G is always no larger than TSA, the number of all s-a pairs. And in most environments in practice, we have $G \ll TSA$. (3) When ϵ is not small or T is very large, the sample complexity has linear dependences on T, S and A since the second term dominates.

O-TempLe *does not necessarily require the number of templates G to be small.* A large G suggests the environment is highly stochastic, e.g., the slipping probabilities of every grid in maze is sampled from a Gaussian distribution. In this case, we can still cluster s-a pairs with adequately close templates, as verified in experiments (see Section 6.5.3).

Proof Sketch. We first show that for any s-a pair, $m_s = \tilde{\mathcal{O}}(\frac{1}{\omega^2})$ samples would guarantee correct template identification and aggregation, and $m = \tilde{\mathcal{O}}(\frac{SV_{\max}^2}{\epsilon^2(1-\gamma)^2})$ samples are sufficient for estimating the s-a transition dynamics. Then we prove that all s-a pairs reach m_s within finite

steps. Finally, by computing the number of visits to unknown s-a pairs and applying the PAC-MDP theorem proposed by [158], we get the sample complexity result. Proof details are in Section 6.7.1.

Comparison with a single-task learner. If RMax is sequentially run for every task, the total sample complexity for T tasks is $\tilde{\mathcal{O}}\left(\frac{TS^2AV_{\max}^3}{\epsilon^3(1-\gamma)^3}\right)$.

(1) When precision is high, i.e., ϵ is small, a significant improvement is achieved, if $\mathcal{O}(SG) \ll \mathcal{O}(TS^2A)$.

(2) When T is large, as long as $\tilde{\mathcal{O}}(\frac{SV_{\max}^2}{\epsilon^2(1-\gamma)^2}) \gg \tilde{\mathcal{O}}(\frac{1}{\omega^2})$, our O-TempLe gains improved sample efficiency.

(3) O-TempLe will not cause negative transfer among tasks. In the worst case, $G = TSA$ (there is no similarity among all s-a transition dynamics) or $\omega^2 = \tilde{\mathcal{O}}(\frac{SV_{\max}^2}{\epsilon^2(1-\gamma)^2})$, O-TempLe has the same-order sample complexity with RMax.

Theorem 55 (**Sample Complexity of FM-TempLe**). *Under the finite-model assumption of there are at most C MDPs for all tasks, for any given $\epsilon > 0, 1 > \delta > 0$, Algorithm 3 on T tasks follows ϵ-optimal policies for all but*

$$\tilde{\mathcal{O}}\Big(\frac{SGV_{\max}^3}{\epsilon^3(1-\gamma)^3} + \frac{T_1SAV_{\max}}{\omega^2\epsilon(1-\gamma)} + \frac{(T-T_1)DC^2V_{\max}}{\omega^2\epsilon(1-\gamma)}\Big) \tag{6.4}$$

steps with probability at least $1-\delta$, where G is the total number of TTs, $T_1 = \Omega(\frac{1}{p_{\min}} \ln \frac{C}{\delta})$ is the number of tasks in the first phase, where $p_{\min}$ is the minimal probability for a task to be drawn from $\mathcal{M}$.

Remark. (1) When C is very large, or $p_{\min}$ is very small, $T_1 \to T$ and FM-TempLe degenerates to O-TempLe. (2) If $DC^2 < SA$ and $T \gg T_1$, FM-TempLe requires fewer samples than O-TempLe.

Comparison with FMRL [136] FM-TempLe has a large improvement over FMRL in most cases. The sample complexity of FMRL for T tasks in our notation is

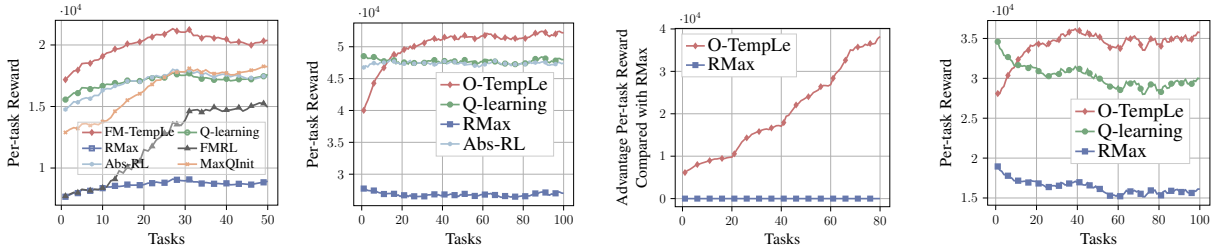
$$\tilde{\mathcal{O}}\Big(\frac{CS^2AV_{\max}^3}{\epsilon^3(1-\gamma)^3}+\frac{T_1S^2AV_{\max}^3}{\epsilon^3(1-\gamma)^3}+(T-T_1)\big(\frac{DC^2V_{\max}}{\Gamma^2\epsilon(1-\gamma)}+\frac{SCV_{\max}^3}{\epsilon^3(1-\gamma)^3}\big)\Big). \tag{6.5}$$

where $T_1 = \Omega(\frac{1}{p_{\min}}\ln\frac{C}{\delta})$, and Γ is the model difference gap defined by [136]. We organize Equation 6.4 and Equation 6.5 both as three-term forms. The first term is for learning of all TTs or all models, where FM-TempLe reduces the dependence on S and gets rid of the dependence on A. The second term is for the first phase, where FMRL performs the same with a single-task RMax learner, while FM-TempLe requires much fewer samples to get optimal policies. Finally, the last term is for the second phase. FMRL needs an additional model elimination step for each task, while FM-TempLe does not. FM-TempLe is worse than FMRL only in extreme cases where there are few MDP models with large model gaps, and a large number of TTs with small TT gaps or ranking gaps.

6.5 Empirical Results

In this section, we demonstrate empirical results to show O-TempLe and FM-TempLe outperform existing state-of-the-art algorithms both in the finite-model setting and in the more realistic online setting. TempLe is able to transfer knowledge between tasks with different sized environments. More importantly, TempLe has a high tolerance to model perturbations; it implements efficient transfer even when the underlying number of TTs is infinite. Our code is available at `https://github.com/umd-huang-lab/template-reinforcement-learning`.

Baselines. We choose the state-of-the-art MTRL algorithms, *Abstraction RL (Abs-RL)* [159], MaxQInit [143] and *FMRL* [136] as baselines. For Abs-RL and MaxQInit, we use the code



(a) Finite-Model MTRL (b) Online MTRL (c) Varying-sized MTRL (d) Infinite number of TTs

Figure 6.3: Performance of O-TempLe and FM-TempLe compared against state-of-the-art baselines in **(a)** Online MTRL (to show TempLe's ability to efficiently transfer knowledge), **(b)** Finite-Model MTRL (to show TempLe outperforms baselines even under environments that the baselines are designed for),**(c)** varying sized MTRL (to show TempLe extends to varying sized state space) and **(d)** Online MTRL with Mixture-of-Gaussians distributed landforms (to show TempLe's robustness against noise and model-perturbation). All results are averaged over 20 different random sequences of tasks. Confidence intervals are omitted to reduce overlapping.

provided by authors. Note that Abs-RL and MaxQInit have multiple versions due to the selection of different base learners, we show the ones with their best performance in this section. Abs-RL works for both the online and finite-model setting, whereas MaxQInit and FMRL work for the finite-model setting only, since they both require the number of tasks to be small and known. Meanwhile, to show the effectiveness of our proposed algorithms and other MTRL algorithms, we also run RMax and Q-learning [160] for every single task without knowledge transfer.

6.5.1 Finite-Model MTRL

Environment. All the baselines including FMRL, Abs-RL, MaxQInit are designed for the finite-model setting (note that Abs-RL also works in the online setting), where the number of models C is small. We use a similar maze environment as in FMRL, where MDPs only differ at the goal state.

Performance. We generate two 4×4 maze tasks with different goal states as the underlying models, and then randomly sample 50 tasks from the two underlying models. Figure 6.3a shows

the comparison of per-task rewards. FMRL has the same performance with RMax in the model-collecting phase, and then achieves increasing rewards in the following tasks after it successfully identifies the underlying two types of MDPs. After 30 tasks, all state-actions pairs in the models become known, so the per-task reward converges. Similarly, MaxQInit gains more rewards when it collects adequate knowledge of the Q values. In contrast, FM-TempLe has a better start as it learns TTs from the beginning. And model identification further helps with efficient learning. Over all tasks, FM-TempLe substantially outperforms other agents, despite that baselines are designed for the finite-model case.

6.5.2 Online MTRL

Environment. For the more realistic Online MTRL which allows the *number of MDP models to be extremely large*, we generalize the traditional maze environment to have arbitrary combinations of landforms, as shown in Figure 6.1. We use 3 types of landforms, sand, marble and ice, respectively with slipping probabilities 0, 0.2, and 0.4. In this scenario, under a certain number of states S, the number of possible tasks is exponential in S.

Performance. In the online setting, we consider 4×4 mazes with different arrangements of landforms streaming in. The per-task rewards of each agent are displayed in Figure 6.3b. Among all agents, our O-TempLe obtains the highest average reward. We see during the first 40 tasks, the performance of O-TempLe continuously and rapidly grows by transferring previous knowledge. In contrast, the performance of Abs-RL does not increase as more tasks come in and keeps the same with single-task Q-learning, because the maze environment is not efficiently abstracted by Abs-RL.

Performance on Varying State Space. To show the feasibility of TempLe for varying-sized environment tasks, and its ability to generalize knowledge learned in small tasks to speed up learning in larger tasks, we vary the size of the mazes across tasks. More specifically, the first 20 tasks are 3×3 mazes, followed by 20 4×4 mazes, 20 5×5 mazes and 20 6×6 mazes. We show O-TempLe's per-task advantage rewards over single task RMax in Figure 6.3c, since other MTRL baselines are not feasible in this setting. The performance advantage over RMax increases over more observed tasks, verifying that O-TempLe transfers knowledge among different-sized mazes. Experiments on varying action spaces are shown in Section 6.7.2.2.

6.5.3 MTRL with Infinite TTs

Environment. We also conduct experiments to show TempLe's robustness to noise and model perturbations. which is crucial for its application to real-world settings where "landforms" could vary continuously. We draw the landforms (slipping probabilities) of each grid from a mixture of Gaussian distributions, which are centered at 0.2, 0.4, and 0.6 with standard derivation 0.05. In this case, the number of TTs could be infinitely large.

Performance. We show O-TempLe's per-task advantage rewards over single task RMax and Q-learning in Figure 6.3d, in which O-TempLe still achieves successful multi-task learning. This result demonstrates O-TempLe's ability of tolerating noise and generalizing to real-life applications.

6.5.4 Robustness to Hyper-parameters

TempLe requires a user-specified TT gap $\hat{\tau}$ as input. Also, both FMRL and FM-TempLe

require a user-specified model gap Γ. We test various hyper-parameters to understand how significantly the performance of the algorithms could be affected by inaccurate guesses of $\hat{\tau}$ and Γ, shown in Figure 6.4.

(a) TT gap $\hat{\tau}$

(b) Model gap Γ

Figure 6.4: Hyper-parameter test of TT gap $\hat{\tau}$ and model gap Γ(the vertical dashed line shows the underlying true value).

According to Figure 6.4a, the performance of O-TempLe drops when $\hat{\tau}$ is too large. However, the rewards remain high for relatively small $\hat{\tau}$. Figure 6.4b shows that FM-TempLe gets higher rewards than RMax when setting $\Gamma \leq 1$, although Γ has a larger influence on FM-TempLe compared to FMRL, potentially because the failure of model clustering will cause more inaccurate TT identification. Note that by definition, both $\hat{\tau}$ and Γ would not exceed 2 (see Lemma 6). So we still have a large chance to get higher rewards than RMax by making an educated guess. The results in Figure 6.4 guide the users to specify hyper-parameters when using TempLe.

6.6 Conclusion

This chapter proposes TempLe, the first PAC-MDP MTRL algorithm that works for tasks with varying state/action space without any inter-task mappings or prior knowledge of the MDP

structures. Two algorithms, O-TempLe and FM-TempLe, are introduced for online setting and finite-model setting respectively. We show in theory and experiments that O-TempLe and FM-TempLe both achieve higher sample efficiency than state-of-the-art methods. This work can be extended in many directions. For example, one may benefit from investigating transition probability and reward separately. Our future work includes extension to continuous MDPs by discretizing the state/action space. It is also possible to apply our idea to deep model-based RL, where the the state prediction is usually a Gaussian distribution, then the learned derivation can be regarded as "templates" and augmented by grouping.

6.7 Supplemental Materials: Proofs and Additional Details

6.7.1 Theoretical Analysis and Proofs

6.7.1.1 Upper Bound of TT Gap

Lemma 56. *Let* $\mathbf{a} = (a_1, \ldots, a_n)$ *and* $\mathbf{b} = (b_1, \ldots, b_n)$ *be two vectors in* $\mathbb{R}^n$ *such that* $\sum_{i=1}^n a_i = \sum_{j=1}^n b_j = 1$. *Moreover, assume there hold*

$$1 \geq a_1 \geq a_2 \geq \cdots \geq a_n \geq 0,$$

$$1 \geq b_1 \geq b_2 \geq \cdots \geq b_n \geq 0.$$

Then

$$||\mathbf{a} - \mathbf{b}||_2^2 := \sum_{i=1}^{n} (a_i - b_i)^2 \leq \frac{n-1}{n}. \tag{6.6}$$

The equality holds when we choose, e.g., $\mathbf{a} = (1, 0, \ldots, 0)$ *and* $\mathbf{b} = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$.

Proof. We prove the lemma by induction.

- When $n = 1$, $\mathbf{a} = \mathbf{b} = 1$. Thus the inequality is trivial.

- Assume (6.6) holds for $n = k$, $k \geq 1$. We show that (6.6) is also true for $n = k + 1$. Given vectors $\mathbf{a} = (a_1, \ldots, a_k, a_{k+1})$ and $\mathbf{b} = (b_1, \ldots, b_k, b_{k+1})$ such that they satisfy the conditions in the lemma, construct two new vectors:

$$\mathbf{a}' := \left(a_1 + \frac{a_{k+1}}{k}, \ldots, a_k + \frac{a_{k+1}}{k}\right),$$
$$\mathbf{b}' := \left(b_1 + \frac{b_{k+1}}{k}, \ldots, b_k + \frac{b_{k+1}}{k}\right).$$

It is obvious that $\mathbf{a}'$ and $\mathbf{b}'$ satisfy the conditions in the induction hypothesis. Thus,

$$||\mathbf{a}' - \mathbf{b}'||_2^2 \leq \frac{k-1}{k}. \tag{6.7}$$

We calculate

$$||\mathbf{a}' - \mathbf{b}'||_2^2 = \sum_{i=1}^{k}(a_i - b_i + \frac{a_{k+1} - b_{k+1}}{k})^2 \tag{6.8}$$
$$= \sum_{i=1}^{k}(a_i - b_i)^2 + \frac{(a_{k+1} - b_{k+1})^2}{k} + \frac{2(a_{k+1} - b_{k+1})}{k} \cdot \sum_{i=1}^{k}(a_i - b_i) \tag{6.9}$$
$$= \sum_{i=1}^{k}(a_i - b_i)^2 - \frac{(a_{k+1} - b_{k+1})^2}{k}, \tag{6.10}$$

where the last equality comes from the fact

$$\sum_{i=1}^{k}(a_i - b_i) = b_{k+1} - a_{k+1}.$$

By assumptions,

$$\sum_{i=1}^{k+1} a_i = 1 \text{ and } a_1 \geq a_2 \geq \cdots \geq a_{k+1} \geq 0,$$

hence

$$a_{k+1} \in \left[0, \frac{1}{k+1}\right]. \tag{6.11}$$

Similarly,

$$b_{k+1} \in \left[0, \frac{1}{k+1}\right]. \tag{6.12}$$

Combine (6.7), (6.10), (6.11) and (6.12),

$$\begin{aligned} ||\mathbf{a}-\mathbf{b}||_2^2 &= \sum_{i=1}^{k+1}(a_i - b_i)^2 \\ &\leq \frac{k-1}{k} + \frac{k+1}{k}(a_{k+1} - b_{k+1})^2 \\ &\leq \frac{k-1}{k} + \frac{k+1}{k} \cdot \frac{1}{(k+1)^2} \\ &\leq \frac{k}{k+1}. \end{aligned}$$

Thus (6.6) also holds for $n = k+1$. The proof is finished.

□

6.7.1.2 Proof of Theorem 54

To prove Theorem 54, we first present Lemma 57, Lemma 59, Lemma 60, Lemma 61 and Lemma 62.

Lemma 57 and Lemma 59 provide the sample size requirements for correctly identifying the TT of an s-a pair.

Lemma 57. *For any state-action pair, suppose the ranking permutation of the estimated probability vector is the same with that of the underlying probability vector, then it would be identified to its corresponding TT group correctly with $\mathcal{O}(\frac{1}{\tau^2}\ln\frac{1}{\delta})$ samples, with probability at least $1-\delta$, where τ is the TT gap defined in Definition 51.*

Proof. For an state-action pair (s,a), define the observation vector of the i^{th} sample as $Z_i = [\mathbb{I}_{s'=s_1}, \mathbb{I}_{s'=s_2}, \cdots, \mathbb{I}_{s'=s_S}, r]$.

Define $X_n = \sum_{i=1}^{n} Z_i - n\theta(s,a)$, and set $X_0 = 0$.

We first prove the sequence $\{X_n\}$ is a vector-valued martingale.

$$\begin{aligned}
\mathrm{E}\left(X_n | X_0, X_1, \cdots, X_{n-1}\right) &= E\left[\sum_{i=1}^{n} Z_i - n\theta(s,a) | X_0, X_1, \ldots, X_{n-1}\right] \\
&= E\left[\sum_{i=1}^{n-1} Z_i - (n-1)\theta(s,a) + Z_n - \theta(s,a) | X_0, X_1, \ldots, X_{n-1}\right] \\
&= X_{n-1} + E[Z_n - \theta(s,a)] \\
&= X_{n-1}
\end{aligned}$$

Obviously, $E[\|X_n\|] < \infty$ for all n. Thus $\{X_n\}$ is a (strong) martingale.

By application of the extended Hoeffding's inequality [161], we get

$$Pr(\|\frac{1}{n}\sum_{i=1}^{n} Z_i - \theta(s,a)\| \geq \epsilon) \leq 2\mathrm{e}^{2-\frac{n\epsilon^2}{2}}$$

Set the failure probability as δ, we obtain $n \geq \mathcal{O}(\frac{1}{\epsilon^2}\ln\frac{1}{\delta})$.

□

Lemma 57 assumes perfect permutation, and Lemma 59 addresses the problem of how to avoid notable corruptions of permutations. For ease of illustrating, we define the concept *almost the same* and *almost correct* in Definition 58.

Definition 58 (Almost the same and almost correct)**.** *If for two probability vectors p and p', their ranking permutations σ and σ' are the same except for elements whose difference is smaller than $\mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$, then we call σ and σ' almost the same. If p' is the approximation of the ground truth p, then we call σ' almost correct.*

Lemma 59. *With $\tilde{\mathcal{O}}(\frac{1}{\nu^2}\ln\frac{S}{\delta})$ samples of a s-a pair, its transition permutation will be almost correct, with probability $1-\delta$.*

Proof. Suppose there is a transition probability vector p with length l, as well as transition-difference gap ν. We estimate p by randomly sampling indices $1, \cdots, i, \cdots, l$ according to the probability distribution p. Let $\hat{p} = [\frac{n(1)}{n}, \cdots, \frac{n(l)}{n}]$.

For any two adjacent elements p_i and p_j (adjacent means there is no p_k whose value is between p_i and p_j), we want our estimations $\frac{n(i)}{n}$ and $\frac{n(j)}{n}$ to satisfy $\frac{n(i)}{n} > \frac{n(j)}{n}$. It is sufficient if we guarantee $\frac{n(i)}{n}$ and $\frac{n(j)}{n}$ are respectively $\nu/2$-close to p_i and p_j. By Hoeffding's inequality,

$$P(|\frac{n(i)}{n} - p_i| > \frac{\nu}{2}) \leq 2\exp(-2n(\nu/2)^2)$$

Therefore, $n \geq \mathcal{O}(\frac{1}{\nu^2}\ln\frac{1}{\delta})$ is sufficient. By union bound, we have $n \geq \mathcal{O}(\frac{1}{\nu^2}\ln\frac{S}{\delta})$. □

Lemma 57 and Lemma 59 imply that the small threshold should satisfy $m_s = \tilde{\mathcal{O}}(\frac{1}{\min\{\tau^2,\nu^2\}})$.

Then, Lemma 60 claims if horizon is set to be large enough, all s-a pairs will have sufficient samples to be correctly grouped.

Lemma 60. *If $H = \tilde{\mathcal{O}}(\frac{DSA}{\omega^2})$, all state-action pairs in the task will have at least $\tilde{\mathcal{O}}(\frac{1}{\omega^2}\ln\frac{TSA}{\delta})$ samples with probability $1-\delta$.*

The proof of Lemma 60 is similar to Lemma 2.1 in paper [136].

Lemma 61 is a variant of the "simulation lemma" [154, 155] with TT estimation.

Lemma 61. *For any two MDPs M and $\tilde{M}$ with the same $\mathcal{S}, \mathcal{A}, \mu, \gamma$, if for any s-a pair (s,a), the ranking permutations of $p(s,a)$ and $\tilde{p}(s,a)$ are almost the same, and $\mathbf{g} = (desc(p(s,a)), r(s,a))$ as well as $\tilde{\mathbf{g}} = (desc(\tilde{p}(s,a)), \tilde{r}(s,a))$ satisfy $\|\mathbf{g}^{(p)} - \tilde{\mathbf{g}}^{(p)}\| \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$ and $|g^{(r)} - \tilde{g}^{(r)}| \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$, then for any policy π, $|V_M^\pi - V_{\tilde{M}}^\pi| \leq \epsilon$.*

Proof. For an s-a pair (s,a), suppose its ranking permutation in M is σ, and the ranking permutation in $\tilde{M}$ is $\tilde{\sigma}$.

We first assume σ and $\tilde{\sigma}$ are exactly the same. So we have $\mathbf{g}^{(p)} = \sigma(p(s,a))$ and $\tilde{\mathbf{g}}^{(p)} = \sigma(\tilde{p}(s,a))$.

Thus $\|\mathbf{g}^{(p)} - \tilde{\mathbf{g}}^{(p)}\| \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$ implies $\|p(s,a) - \tilde{p}(s,a)\| \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$, because of the property of permutation.

Similarly, $|g^{(r)} - \tilde{g}^{(r)}| \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$ implies $|r(s,a) - \tilde{r}(s,a)| \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$.

Then, following the standard proof [162, 163], it is easy to show $|V_M^\pi - V_{\tilde{M}}^\pi| \leq \epsilon$.

Next, we allow σ and $\tilde{\sigma}$ be almost the same (see Definition 58), and show that it only causes up to a constant factor increase in the value difference $|V_M^\pi - V_{\tilde{M}}^\pi|$.

Without loss of generality, assume $\tilde{\sigma}$ only reverses σ in indices i and j, i.e., $\sigma(i) = \tilde{\sigma}(j)$ and $\sigma(j) = \tilde{\sigma}(i)$. According to the definition of almost the same, $p_{\sigma(i)} - p_{\sigma(j)} = p_{\tilde{\sigma}(j)} - p_{\tilde{\sigma}(i)} \leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$ Then we have

$$
\begin{aligned}
\|p(s,a) - \tilde{p}(s,a)\| &= \|\sigma^{-1}(\mathbf{g}^{(p)}) - \tilde{\sigma}^{-1}(\tilde{\mathbf{g}}^{(p)})\| \\
&= |g_1^{(p)} - \tilde{g}_1^{(p)}| + \cdots + |g_i^{(p)} - \tilde{g}_j^{(p)}| + |g_j^{(p)} - \tilde{g}_i^{(p)}| + \cdots + |g_S^{(p)} - \tilde{g}_S^{(p)}| \\
&\leq \|\mathbf{g}^{(p)} - \tilde{\mathbf{g}}^{(p)}\| + |g_i^{(p)} - \tilde{g}_j^{(p)}| + |g_j^{(p)} - \tilde{g}_i^{(p)}| \\
&\leq \mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}}) + \mathcal{O}(\frac{2\epsilon(1-\gamma)}{\sqrt{S}V_{\max}}) \\
&\leq (1 + \frac{2}{\sqrt{S}})\mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})
\end{aligned}
$$

Therefore, if $\tilde{\sigma}$ differs with σ in all indices, as long as they are almost the same, $|V_M^\pi - V_{\tilde{M}}^\pi| \leq 2\epsilon$. By adjusting the constant factor, $|V_M^\pi - V_{\tilde{M}}^\pi| \leq \epsilon$ also holds. □

According to Lemma 61, if each TT gets $\mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$-accurate estimation, then all the s-a transition dynamics associated with the same TT will be accurate enough to generate an ϵ-optimal policy. Therefore, the regular known threshold m is still the same as in RMax, i.e.,

$m = \tilde{\mathcal{O}}(\frac{SV_{\max}^2}{\epsilon^2(1-\gamma)^2})$. Note that the small threshold should not exceed the regular threshold, so $m_s = \tilde{\mathcal{O}}(\frac{1}{\omega^2})$, where $\omega = \max\{\min(\tau, \nu), \mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})\}$. If τ or ν is smaller than $\mathcal{O}(\frac{\epsilon(1-\gamma)}{\sqrt{S}V_{\max}})$, then the small threshold becomes the regular threshold and O-TempLe degenerates to RMax.

Next, we show in Lemma 62 the total number of visits to unknown state-action pairs during T tasks.

Lemma 62. *The total number of visits to unknown s-a pairs during the execution of Algorithm 11 for T tasks is*

$$\tilde{\mathcal{O}}\Big(\frac{TSA}{\omega^2} + \frac{SGV_{\max}^2}{\epsilon^2(1-\gamma)^2}\Big) \tag{6.13}$$

Proof. For every task, Algorithm 11 first uses known threshold $m_s = \tilde{\mathcal{O}}(\frac{1}{\omega^2})$ for all s-a pairs. And the first m_s visits to an s-a pair are all visits to unknowns. So all the SA s-a pairs over T tasks take $\mathcal{O}(\frac{TSA}{\omega^2})$ steps of visiting unknowns in total.

Once an s-a pair is roughly known (having visits more than m_s), the TT is identified, and the known threshold is changed to m for the s-a pair. If the corresponding TT is fully known (having visits more than m), then the s-a pair immediately becomes fully known by incorporating all visit counts of the TT. If the corresponding TT is not fully known yet, visits to the s-a pair are still counted as visits to unknown, until the TT is known. Therefore, for every possible TT, there are m unknown visits. And G TTs result in Gm unknown visits, which is the second term in Equation 6.13 □

Now we can proceed to prove the main theorem.

Proof. (of Theorem 54) We apply the PAC-MDP theorem proposed by [163] to get the sample complexity of O-TempLe. Proposition 1 in [163] claims that any greedy learning algorithm with

known set K and known state-action MDP M_K satisfies 3 conditions (optimism, accuracy and learning complexity) will follow a 4ϵ-optimal policy on all but $\mathcal{O}\left(\frac{\zeta(\epsilon,\delta)}{\epsilon(1-\gamma)}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$ timesteps with probability $1-2\delta$, where $\zeta(\epsilon,\delta)$ is the total number of updates of action-value estimates plus the number of visits to unknowns. This proposition, while it focuses on single-task learners, can be easily adapted to work for multi-task learners, as shown in [136].

Now we verify that the required 3 conditions all hold for our algorithm.

(1) $Q_t(s,a) \geq Q^*(s,a) - \epsilon$ for any timestep t (optimism).

This condition naturally holds as the single-task learner RMax chooses actions by optimistic value functions. O-TempLe does not change the way of choosing actions. It is similar for using E^3 or MBIE as the single-task learner.

(2) $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$ for any timestep t (accuracy).

An s-a pair is in M_K if it is fully known, i.e., $n(s,a) \geq m$. A part of $n(s,a)$ may come from the visits to other s-a pairs with the same TT. According to Lemma 61, condition (2) holds if the estimation of the TT is within $\mathcal{O}(\frac{\epsilon(1-\gamma)}{V_{\max}})$ accuracy. By Hoeffding's inequality, to achieve this accuracy, $m = \tilde{\mathcal{O}}(\frac{SV^2_{\max}}{\epsilon^2(1-\gamma)^2})$ samples are required for a TT.

(3) The total number of updates of action-value estimates plus the number of visits to unknowns is bounded by $\zeta(\epsilon,\delta)$ (learning complexity).

Lemma 62 already gives the number of visits to unknown s-a pairs, and the updates of action-value estimates will happen no more than TSA times for T tasks. Hence, $\zeta(\epsilon,\delta) = \tilde{\mathcal{O}}\left(\frac{TSA}{\omega^2} + \frac{SGV^2_{\max}}{\epsilon^2(1-\gamma)^2}\right)$.

Therefore, the sample complexity of O-TempLe is

$$\tilde{\mathcal{O}}\left(\left(\frac{TSA}{\omega^2} + \frac{SGV^2_{\max}}{\epsilon^2(1-\gamma)^2}\right)\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)\right)$$

□

6.7.1.3 Proof of Theorem 55

Proof. (of Theorem 55)

The proof of Theorem 55 is similar to the proof of Theorem 54, because FM-TempLe is adapted from O-TempLe. The only difference lies in the number of visits to unknown s-a pairs.

In the first phase, FM-TempLe is the same with O-TempLe, so the number of visits to identify TTs is $\tilde{\mathcal{O}}(\frac{T_1 SA}{\omega^2})$.

In the second phase, FM-TempLe avoids visiting all s-a pairs for at least m_s times under the help of finite models. As [136] shows, we need at most C^2 informative s-a pairs to fully identify a model, where an s-a pair is "informative" if at least two MDP models have sufficient disagreement in its dynamics. Similarly with Lemma 60, $\tilde{\mathcal{O}}(\frac{DC^2}{\omega^2})$ samples are enough to let all these C^2 informative s-a pairs roughly known. Then the correct model for the current task would be identified. Thus, for every task in the second phase, $\tilde{\mathcal{O}}(\frac{DC^2}{\omega^2})$ visits to unknowns are needed.

Finally, for each TT, its visits are shared among s-a pairs and tasks, no matter which phase they are in. Hence there are still $\tilde{\mathcal{O}}(\frac{SGV_{\max}^3}{\omega^2 \epsilon(1-\gamma)})$ visits to unknowns.

Adding the above three parts of visits to unknowns, and following the proof of Theorem 54, we obtain the sample complexity of Theorem 55.

□

6.7.2 Experiment Settings and Additional Results

6.7.2.1 Setups

Computing Infrastructure All experiments are conducted on a PC equipped with a 3.6 GHz INTEL CPU of 6 cores.

Hyper-parameter Settings In Maze, an agent navigates with actions "up", "down", "left" and "right". The reward of the goal state is set to be 1.0, and the step cost is set as 0.2.

The base learners in FMRL, our O-TempLe and our FM-TempLe are chosen to be RMax (known threshold being 500) without loss of generality. The threshold m_s is set to be 50, the number of episodes 3000, and the number of in-episode steps 30. $\hat{\tau}$ is set to be 0.15 for online MTRL environments, and 0.24 for Finite-Model environments. Model gap Γ for FMRL and FM-TempLe is set to be 0.6. In Finite-Model MTRL experiments, T_1 is set to be 15 for both FM-TempLe and FMRL.

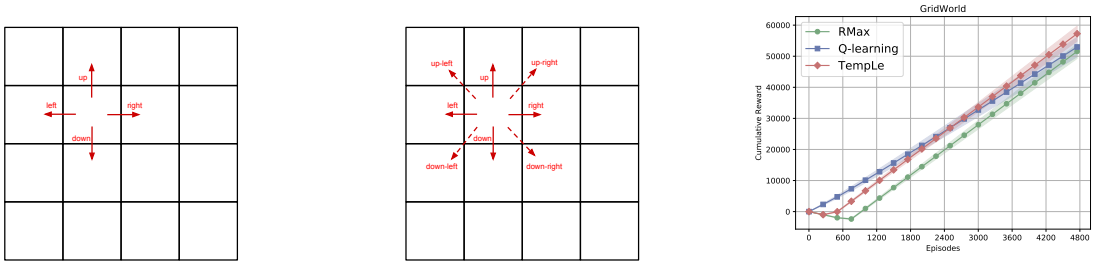
The results are averaged over 20 runs. The randomization in the multiple runs comes from different task sequences generated across runs, although the comparison in each run is done on the same task sequence. We also provide the generated task sequences in our code to ensure reproducibility.

6.7.2.2 Results of Varying Action Size

Our proposed method can also work when the action space of the tasks are different, which could happen in transfer reinforcement learning (TRL) settings. For example, in a navigating task, the available actions can be simply "up", "down", "left" and "right" (as shown in Fig-

ure 6.5a), but a more difficult task may also allow the actions "up-left", "up-right", "down-left", "down-right" (as shown in Figure 6.5b). Intuitively, there is some shared knowledge between these two tasks, and the agent will learn the 8-action task better if it can transfer some knowledge from the 4-action task. However, few existing methods can transfer appropriate knowledge between these two tasks without pre-defined inter-task mappings. In contrast, our proposed TempLe is able to transfer knowledge between these two tasks without any prior knowledge.

In Figure 6.5c, we show the performance of Q-learning, RMax and TempLe on the 8-action gridworld, where TempLe has already learned from a 4-action task and gathered the TT information. The results suggest that TempLe automatically figures out the relation among the state-action pairs in two tasks with different action spaces.



(a) Source task (b) Target task (c) CartPole

Figure 6.5: Transfer learning with varying action size.

6.7.2.3 Universal Applicability of TempLe

We also observe that our proposed template learning is universally applicable to many classical stochastic environments. For example, all gridworld-based environments like FourRoom, Taxi, FrozenLake [164], etc. In addition, Strehl et al. [165] propose 3 challenging MDPs as Figure 6.6 shows. It can be seen from these MDP definition that the number of templates is smaller than the number of state-action pairs in all of them. For instance, the TT $((1, 0, \cdots, 0), 0)$ ap-

pears for multiple times in all of the three tasks. Thus, in each of the environments, TempLe can transfer knowledge from known state-action pairs to unknown state-action pairs with the same TT. More interestingly, since these tasks have some common TTs, if we sequentially learn these three tasks, TempLe has the potential to transfer knowledge among them, in spite that they are totally different environments in common sense.

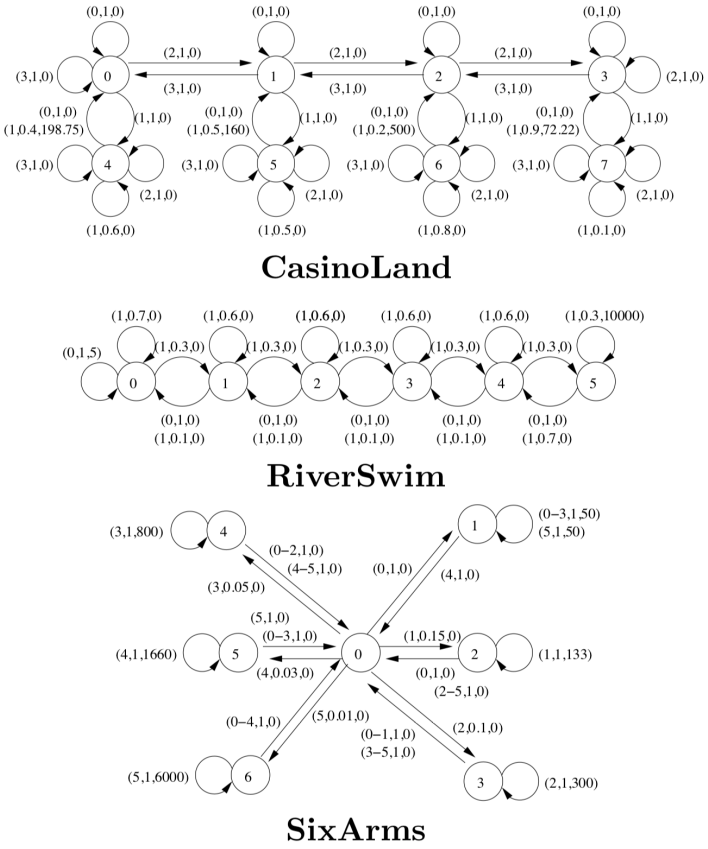
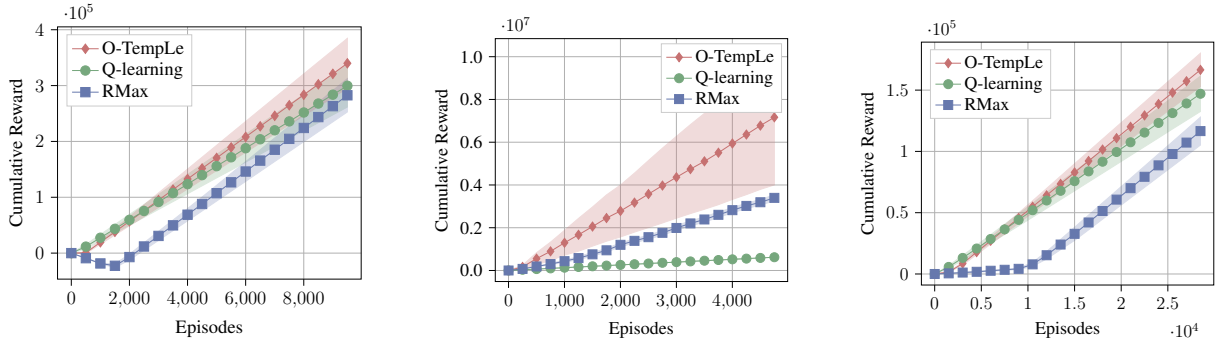


Figure 6.6: **Three challenging MDPs [165]**: CasinoLand (top), RiverSwim (middle), and SixArms (bottom). Each node in the graph is a state and each edge is labeled by one or more transitions. A transition is of the form (a, p, r) where a is an action, p the probability that action will result in the corresponding transition, and r is the reward for taking the transition.

Figure 6.7b Figure 6.7a, and Figure 6.7c respectively show the performance of TempLe compared with baselines on RiverSwim[165], FourRoom and a large GridWorld, which are well-known hard-to-explore environments. TempLe outperforms the single-task learners, because it

can aggregate similar information in the environment, which saves samples and facilitates exploration.



(a) 9×9 FourRoom (b) RiverSwim (c) 10×10 GridWord

Figure 6.7: Performance of O-TempLe on challenging single-task problems, compared with RMax and Q-learning.

6.7.3 Discussion: Potential Extension to Deep RL

Model-based deep RL is an important research area [166, 167], where the learner learns a dynamics model of the environment. More specifically, the learner attempts to learn a function f (usually parameterized by a neural network θ) such that $f_\theta(s_t)$ approximates s_{t+1}, where s_t is the current state and s_{t+1} is the next state. The reward function can be modeled in a similar way, while we only discuss the transition model here for simplicity.

Our proposed TempLe can be extended to large-scale MDPs and deep RL to learn the dynamics model more accurately. Below we explain the concrete method and some empirical results.

TempLe is essentially estimating the "relative" transition among states due to the permutation operation. For example, TempLe considers the transition from s_1 to s_2 with probability 0.5 to be similar to the transition from s_7 to s_8 with probability 0.5, since the relative state difference of

them is the same. This is equivalent to predicting a "state shift" in a continuous state space, which is $s_{t+1} - s_t$. In our paper, we focus on discrete state space and model the transition probabilities with discrete distributions. Similarly, in the continuous case, we can use continuous distributions (e.g. Gaussian) to approximately model the state shift, without doing state counting and ranking. Note that Nagabandi et al. [167] also use the relative state shift in their deep RL model, whose experiments have justified the advantages of using relative state shift rather than absolute state difference. But their model is deterministic while we consider stochastic cases.

In addition to the relative state shift modeling, another key idea of TempLe is to cluster the old state-action dynamics and augment the new state-action dynamics. In the continuous case, if we assume the transition probabilities are from a mixture of Gaussian distributions, then a similar cluster and augment method can also be used. The extended algorithm works as follows:

(1) use a neural network (NN) to predict the relative state shift: $\hat{\Delta} \approx s_{t+1} - s_t$;

(2) approximately model Δ's using a mixture of Gaussian (other distribution models are also applicable). From the trajectories/history, we compute $\Delta = s_{t+1} - s_t$, cluster them (GEN-TT/TT-UPDATE step of TempLe) and use the averaged $\bar{\Delta}$ from each Gaussian subpopulation/cluster to improve the prediction of the NN by minimizing MSE($\hat{\Delta}$, $\bar{\Delta}$);

(3) use $\bar{\Delta}$ to augment the accuracy of $\hat{\Delta}$ by identifying it into an existing cluster (AUGMENT step in TempLe). As a result, we can learn an accurate prediction model of the environment.

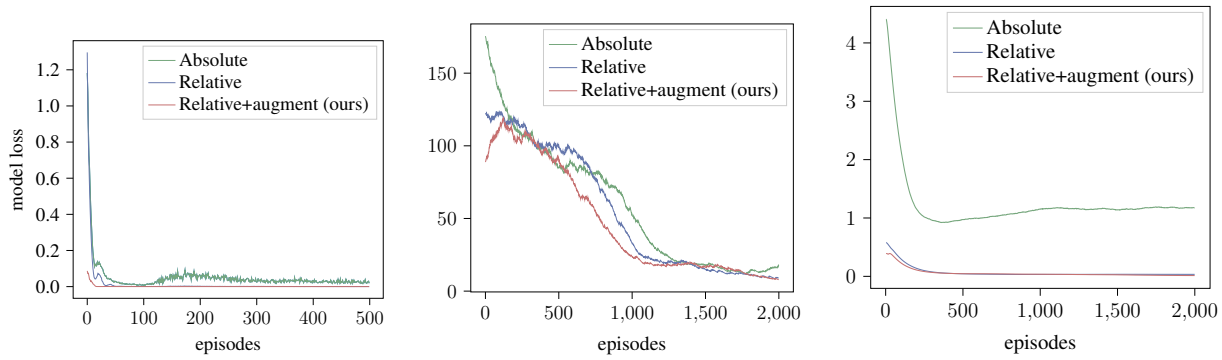
We implemented the above idea on the continuous environments CartPole, LunarLander and Mujoco Hopper. We use a 2-layer MLP with 64 nodes per layer. The model learning methods are summarized as below.

- **Absolute.** Directly predict the absolute next state.

- **Relative.** Predict the relative state shift [167].

- **Relative+augment (ours).** Predict the relative state shift, and augment the model by clustering.

The results are shown in Figure 6.8, where we can see that our method learns the most accurate model compared with two baselines, because learning relative state shift reduces the variance and the augmentation allows knowledge transferring among state-actions with similar dynamics.



(a) CartPole (b) LunarLander (c) Hopper

Figure 6.8: Extending TempLe to deep RL.

Chapter 7: Adapting to Novel Observation Feature Spaces

7.1 Introduction

Deep Reinforcement Learning (DRL) has the potential to be used in many large-scale applications such as robotics, gaming and automotive. In these real-life scenarios, it is an essential ability for agents to utilize the knowledge learned in past tasks to facilitate learning in unseen tasks, which is known as Transfer RL (TRL). Most existing TRL works [138, 168] focus on tasks with the same state-action space but different dynamics/reward. However, these approaches do not apply to the case where the observation space changes significantly.

Observation change is common in practice as in the following scenarios. (1) Incremental environment development. RL is used to train non-player characters (NPC) in games [169], which may be frequently updated. When there are new scenes, characters, or obstacles added to the game, the agent's observation space will change accordingly. (2) Hardware upgrade/replacement. For robots with sensory observations [170], the observation space could change (e.g. from text to audio, from lidar to camera) as the sensor changes. (3) Restricted data access. In some RL applications [171], agent observation contains sensitive data (e.g. inventory) which may become unavailable in the future due to data restrictions. In these cases, the learner may have to discard the old policy and train a new policy from scratch, as the policy has a significantly different input space, even though the underlying dynamics are similar. But training an RL policy from scratch

can be expensive and unstable. Therefore, there is a crucial need for a technique that transfers knowledge across tasks with similar dynamics but different observation spaces.

Besides these existing common applications, there are more benefits of across-observation transfer. For example, observations in real-world environments are usually rich and redundant, so that directly learning a policy is hard and expensive. If we can transfer knowledge from low-dimensional and informative vector observations (usually available in a simulator) to richer observations, the learning efficiency can be significantly improved. Therefore, an effective transfer learning method enables many novel and interesting applications, such as curriculum learning via observation design.

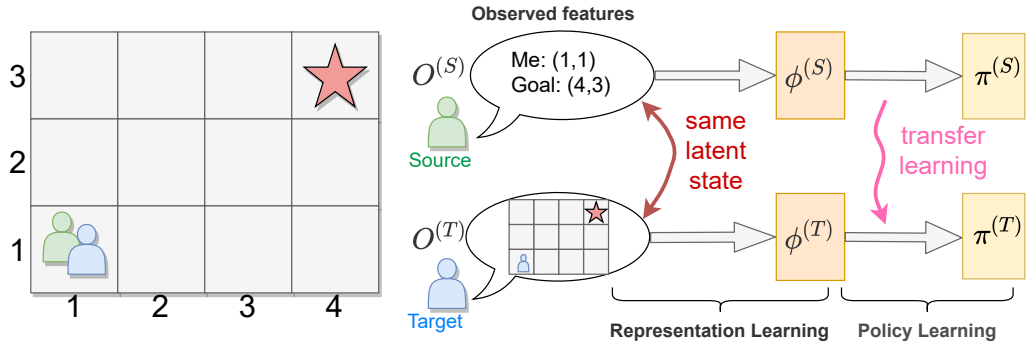


Figure 7.1: An example of the transfer problem with changed observation space. The source-task agent observes the x-y coordinates of itself and the goal, while the target-task agent observes a top-down view/image of the whole maze. The two observation spaces are drastically different, but the two tasks are structurally similar. Our goal is to transfer knowledge from the source task to accelerate learning in the target task, without knowing or learning any inter-task mapping.

In this chapter, we aim to fill the gap and propose a new algorithm that can automatically transfer knowledge from the old environment to facilitate learning in a new environment with a (drastically) different observation space. In order to meet more practical needs, we focus on the challenging setting where the observation change is: **(1) unpredictable** (there is no prior knowledge about how the observations change), **(2) drastic** (the source and target tasks have

significantly different observation feature spaces, e.g., vector to image), and **(3) irretrievable** (once the change happens, it is impossible to query the source task, so that the agent can not interact with both environments simultaneously). Note that different from many prior works [172, 173], we do not assume the knowledge of any inter-task mapping. That is, the agent does not know which new observation feature is corresponding the which old observation feature.

To remedy the above challenges and achieve knowledge transfer, we make a key observation that, if only the observation features change, the source and target tasks share the same latent space and dynamics (e.g. in Figure 7.1, $\mathcal{S}^{(S)}$ and $\mathcal{S}^{(T)}$ can be associated to the same latent state). Therefore, we first disentangle representation learning from policy learning, and then accelerate the target-task agent by regularizing the representation learning process with the latent dynamics model learned in the source task. We show by theoretical analysis and empirical evaluation that the target task can be learned more efficiently with our proposed transfer learning method than from scratch.

Summary of Contributions.

(1) To the best of our knowledge, we are the first to discuss the transfer problem where the source and target tasks have drastically different observation feature spaces, and there is no prior knowledge of an inter-task mapping.

(2) We theoretically characterize what constitutes a "good representation" and analyze the sufficient conditions the representation should satisfy.

(3) Theoretical analysis shows that a model-based regularizer enables efficient representation learning in the target task. Based on this, we propose a novel algorithm that automatically transfers knowledge across observation representations.

(4) Experiments in 7 environments show that our proposed algorithm significantly improves the

learning performance of RL agents in the target task.

7.2 Related Work

Transfer RL across Observation Feature Spaces. Transferring knowledge between tasks with different observation spaces has been studied for years. Many existing approaches[172, 173, 174] require an explicit mapping between the source and target observation spaces, which may be hard to obtain in practice. Raiman et al. [175] introduce network surgery that deals with the change in the input features by determining which components of a neural network model should be transferred and which require retraining. However, it requires knowledge of the input feature maps, and is not designed for drastic changes, e.g. vector to pixel. Sun et al. [12] propose a provably sample-efficient transfer learning algorithm that works for different observation spaces without knowing any inter-task mapping, but the algorithm is mainly designed for tabular RL and model-based RL which uses the model to plan for a policy, different from our setting. Gupta et al. [176] achieve transfer learning between two different tasks by learning an invariant feature space, with a key time-based alignment assumption. We empirically compared this method with our proposed transfer algorithm in Section 7.5. Our work is also related to state abstraction in block MDPs, as studied by Zhang et al. [177]. But the problem studied in [177] is a multi-task setting where the agent aims to learn generalizable abstract states from a series of tasks. Another related topic is domain adaptation in RL [178, 179, 180], where the target observation space (e.g. real world) is different from the source observation (e.g. simulator). However, domain adaptation does not assume drastic observation changes (e.g. changed dimension). Moreover, the aim of domain adaptation is usually zero-shot generalization to new observations, thus prior

knowledge or a few samples of the target domain is often needed [179].

Representation Learning in RL. In environments with rich observations, representation learning is crucial for the efficiency of RL methods. Learning unsupervised auxiliary tasks [181] is shown to be effective for learning a good representation. The relationship between learning policy-dependent auxiliary tasks and learning good representations has been studied in some prior works [182, 183, 184], while our focus is to learn policy-independent auxiliary tasks to facilitate transfer learning. Using latent prediction models to regularize representation has been shown to be effective for various types of rich observations [185, 186]. Gelada et al. [187] theoretically justify that learning latent dynamics model guarantees the quality of the learned representation, while we further characterize the relationship between representation and learning performance, and we utilize dynamics models to improve transfer learning. Zhang et al. [180] use a bisimulation metric to learn latent representations that are invariant to task-irrelevant details in observation. As pointed out by [188], invariant and sufficient representation is indeed minimal sufficient, so it is an interesting future direction to combine our method with bisimulation metric to learn minimal sufficient representations. There is also a line of work using contrastive learning to train an encoder for pixel observations [189, 190, 191], which usually pre-train an encoder based on image samples using self-supervised learning. However, environment dynamics are usually not considered during pre-training. Our algorithm can be combined with these contrastive learning approaches to further improve learning performance in the target task.

7.3 Background and Problem Setup

Representation Learning in RL. Real-world applications usually have large observation spaces for which function approximation is needed to learn the value or the policy. However, directly learning a policy over the entire observation space could be difficult, as there is usually redundant information in the observation inputs. A common solution is to map the large-scale observation into a smaller representation space via a *non-linear encoder* (also called a *representation mapping*) $\phi : \mathcal{S} \to \mathbb{R}^d$, where d is the representation dimension, and then learn the policy/value function over the representation space $\phi(\mathcal{S})$. In DRL, the encoder and the policy/value are usually jointly learned.

7.3.1 Transfer Across Different Observation Spaces

We aim to transfer knowledge learned from a source MDP to a target MDP, whose observation spaces are different while dynamics are structurally similar. Denote the source MDP as $\mathcal{M}^{(S)} = \langle \mathcal{S}^{(S)}, \mathcal{A}, P^{(S)}, R^{(S)}, \gamma \rangle$, and the target MDP as $\mathcal{M}^{(T)} = \langle \mathcal{S}^{(T)}, \mathcal{A}, P^{(T)}, R^{(T)}, \gamma \rangle$. Note that $\mathcal{S}^{(S)}$ and $\mathcal{S}^{(T)}$ can be significantly different, such as $\mathcal{S}^{(S)}$ being a low-dimensional vector space and $\mathcal{S}^{(T)}$ being a high-dimensional pixel space, which is challenging for policy transfer since the source target policy have different input shapes and would typically be very different architecturally.

In this work, as motivated in the Introduction, we focus on the setting wherein the dynamics ($(P^{(S)}, R^{(S)})$ and $(P^{(T)}, R^{(T)})$) of the two MDPs between which we transfer knowledge are defined on different observation spaces but share structural similarities. Specifically, we make the assumption that there exists a mapping between the source and target observation spaces such that the transition dynamics under the mapping in the target task share the same transition dynamics

as in the source task. We formalize this in Assumption 2:

Assumption 2. *There exists a function* $f : \mathcal{S}^{(T)} \to \mathcal{S}^{(S)}$ *such that* $\forall o_i^{(T)}, o_j^{(T)} \in \mathcal{S}^{(T)}, \forall a \in \mathcal{A}$,

$$P^{(T)}(o_j^{(T)}|o_i^{(T)}, a) = P^{(S)}(f(o_j^{(T)})|f(o_i^{(T)}), a), \quad R^{(T)}(o_i^{(T)}, a) = R^{(S)}(f(o_i^{(T)}), a).$$

Remarks. (1) Assumption 2 is mild as many real-world scenarios fall under this assumption. For instance, when upgrading the cameras of a patrol robot to have higher resolutions, such a mapping f can be a down-sampling function. (2) f is a general function without extra restrictions. f can be a many-to-one mapping, i.e., more than one target observations can be related to the same observation in the source task. f can be non-surjective, i.e., there could exist source observations that do not correspond to any target observation.

Many prior works [173, 174] have similar assumptions, but require prior knowledge of such an inter-task mapping to achieve knowledge transfer. However, such a mapping might not be available in practice. As an alternative, we propose a novel transfer algorithm in the next section that *does not* assume any prior knowledge of the mapping f. The proposed algorithm learns a latent representation of the observations and a dynamics model in this latent space, and then the dynamics model is transferred to speed up learning in the target task.

7.4 Proposed Approach

In this section, we first formally characterize *"what a good representation is for RL"* in Section 7.4.1, then introduce our proposed transfer algorithm based on representation regularization in Section 7.4.2, and next provide theoretical analysis of the algorithm in Section 7.4.3.

7.4.1 Characterizing Conditions for Good Representations

As discussed in Section 7.3, real-world applications usually have rich and redundant observations, where learning a good representation [181, 183] is essential for efficiently finding an optimal policy. However, the properties that constitute a good representation for an RL task are still an open question. Some prior works [182, 183, 187] have discussed the representation quality in DRL, but we take a different perspective and focus on characterizing the sufficient properties of representation for learning a task.

Given a representation mapping ϕ, the Q value of any $(o, a) \in \mathcal{S} \times \mathcal{A}$ can be approximately represented by a function of $\phi(o)$, i.e., $\hat{Q}(o, a) = h(\phi(o); \theta_a)$, where h is a function parameterized by θ_a. To study the relation between representation quality and approximation quality, we define an *approximation operator* $\mathcal{H}_\phi$, which finds the best Q-value approximation based on ϕ. Formally, let Θ denote the parameter space of function $h \in \mathcal{H}$, then $\forall a \in \mathcal{A}$, $\mathcal{H}_\phi Q(o, a) := h(\phi(o); \theta_a^*)$, where $\theta_a^* = \text{argmin}_{\theta \in \Theta} \mathbb{E}_o[\|h(\phi(o); \theta) - Q(\phi(o), a)\|]$. Such a function h can be realized by neural networks as universal function approximators [192]. Therefore, the value approximation error $\|Q - \mathcal{H}_\phi Q\|$ only depends on the representation quality, i.e., whether we can represent the Q value of any state o as a function of the encoded state $\phi(o)$.

The quality of the encoder ϕ is crucial for learning an accurate value function or learning a good policy. The ideal encoder ϕ should discard irrelevant information in the raw observation but keep essential information. In supervised or self-supervised representation learning [188, 193], it is believed that a good representation $\phi(X)$ of input X should contain minimal information of X which maintaining sufficient information for predicting the label Y. However, in RL, it is difficult to identify whether a representation is sufficient, since there is no label corresponding

to each input. The focus of an agent is to estimate the value of each input $o \in \mathcal{S}$, which is associated with some policy. Therefore, we point out that the representation quality in RL is *policy-dependent*. Below, we formally characterize the sufficiency of a representation mapping in terms of a fixed policy and learning a task.

Sufficient Representation for A Fixed Policy. We slightly abuse notation and use $\mathcal{H}_\phi$ to denote the approximation operator for state value function $V : \mathcal{S} \to \mathbb{R}$, similar to the approximation of Q as introduced in Section 7.4.1. The following definition characterizes the sufficiency of a representation mapping in terms of evaluating a fixed policy.

Definition 63 (Sufficient Representation for A Fixed Policy)**.** *A representation mapping ϕ is **sufficient** for a policy π w.r.t. approximation operator $\mathcal{H}_\phi$ iff $\mathcal{H}_\phi V^\pi = V^\pi$. More generally, for a constant $\epsilon \geq 0$, Φ is **ϵ-sufficient** for π iff $\|\mathcal{H}_\phi V^\pi - V^\pi\| \leq \epsilon$.*

Remarks. (1) If $o_1, o_2 \in \mathcal{S}$ have different values under π, a good representation should be able to distinguish them, i.e., $\phi^*(o_1) \neq \phi^*(o_2)$.

(2) The approximation operator $\mathcal{H}_\phi$ is linear if $\mathcal{H}_\phi V = \text{Proj}_\Phi(V)$ where $\text{Proj}_\Phi(\cdot)$ denotes the orthogonal projection to the subspace spanned by the basis functions of $\langle \phi_1, \phi_2, \cdots, \phi_d \rangle$.

Sufficient Representation for Learning A Task. The goal of RL is to find an optimal policy. Therefore, it is not adequate for the representation to only fit one policy. Intuitively, a representation mapping is sufficient for learning if we are able to find an optimal policy over the representation space $\phi(\mathcal{S})$, which requires multiple iterations of policy evaluation and policy improvement. Definition 64 below defines a set of "important" policies for learning with $\phi(\mathcal{S})$.

Definition 64 (Encoded Deterministic Policies)**.** *For a given representation mapping $\phi(\cdot)$, define an encoded deterministic policy set Π_ϕ^D as the set of policies that are deterministic and take the*

same actions for observations with the same representations. Formally,

$$\Pi_\phi^D := \{\pi \in \Pi \quad | \quad \exists \tilde{\pi} : \phi(\mathcal{S}) \to \mathcal{A} \text{ s.t. } \forall o \in \mathcal{S}, \pi(o) = \tilde{\pi}(\phi(o))\}, \tag{7.1}$$

where $\tilde{\pi}$ is a mapping from the representation space to the action space.

A policy π is in Π_ϕ^D if it does not distinguish o_1 and o_2 when $\phi(o_1) = \phi(o_2)$. Therefore, Π_ϕ^D can be regarded as deterministic policies that make decisions for encoded observations. Now, we define the concept of sufficient representation for learning in an MDP.

Definition 65 (Sufficient Representation for Learning)**.** *A representation mapping ϕ is **sufficient** for a task $\mathcal{M}$ w.r.t. approximation operator $\mathcal{H}_\phi$ if $\mathcal{H}_\phi Q_\pi = Q_\pi$ for all $\pi \in \Pi_\phi^D$. Furthermore,*

- *ϕ is **linearly-sufficient** for learning $\mathcal{M}$ if $\exists \theta_a$ s.t. $Q_\pi(o, a) = \phi(o)^\top \theta_a$, $\forall a \in \mathcal{A}, \pi \in \Pi_\phi^D$.*
- *ϕ is **ϵ-sufficient** for learning $\mathcal{M}$ if $\|\mathcal{H}_\phi Q_\pi - Q_\pi\| \leq \epsilon$, $\forall \pi \in \Pi_\phi^D$.*

Definition 65 suggests that the representation is sufficient for learning a task as long as it is sufficient for policies in Π_ϕ^D. Then, the lemma below justifies that a nearly sufficient representation can ensure that approximate policy iteration converges to a near-optimal solution.

Lemma 66 (Error Bound for Approximate Policy Iteration)**.** *If ϕ is ϵ-sufficient for task $\mathcal{M}$ (with ℓ_∞ norm), then the approximated policy iteration with approximation operator $\mathcal{H}_\phi$ starting from any initial policy that is encoded by ϕ ($\pi_0 \in \Pi_\phi^D$) satisfies*

$$\limsup_{k \to \infty} \|Q^* - Q^{\pi_k}\|_\infty \leq \frac{2\gamma^2 \epsilon}{(1-\gamma)^2}, \tag{7.2}$$

where π_k is the policy in the k-th iteration.

Lemma 66, proved in Section 7.7.1.1, is extended from the error bound provided by [194]. For simplicity, we consider the bound in ℓ_∞, but tighter bounds can be derived with other norms [195], although a tighter bound is not the focus of this chapter.

How Can We Learn A Sufficient Representation? So far we have provided a principle to define whether a given representation is sufficient for learning. In DRL, the representation is learned together with the policy or value function using neural networks, but the quality of the representation may be poor [183], which makes it hard for the agent to find an optimal policy. Based on Definition 65, a natural method to learn a good representation is to let the representation fit as many policy values as possible as auxiliary tasks, which matches the ideas in other works. For example, Bellemare et al. [182] propose to fit a set of representative policies (called *adversarial value functions*). Dabney et al. [183] choose to fit the values of all past policies (along the *value improvement path*), which requires less computational resource. Different from these works that directly fit the value functions of multiple policies, in Section 7.4.2, we propose to *fit and transfer an auxiliary policy-independent dynamics model*, which is an efficient way to achieve sufficient representation for learning and knowledge transfer, as theoretically justified in Section 7.4.3.

Algorithm 13: Source Task Learning

Input: Regularization weight λ; update frequency m for stable encoder.
Initialize encoder $\phi^{(S)}$, stable encoder $\hat{\phi}^{(S)}$, policy $\pi^{(S)}$, transition prediction network $\hat{P}$ and reward prediction network $\hat{R}$.
for $t = 0, 1, \cdots$ **do**
  Take action $a_t \sim \pi^{(S)}(\phi^{(S)}(o_t^{(S)}))$, get next observation $o_{t+1}^{(S)}$ and reward r_t, store to buffer.
  Sample a mini-batch $\{o_i, a_i, r_i, o_i'\}_{i=1}^N$ from the buffer.
  Update $\hat{P}$ and $\hat{R}$ using one-step gradient descent with $\nabla_{\hat{P}} L_P(\hat{\phi}^{(S)}; \hat{P})$ and $\nabla_{\hat{R}} L_R(\hat{\phi}^{(S)}; \hat{R})$, where L_P and L_R are defined in Equation (7.3).
  Update encoder and policy by $\min_{\pi^{(S)}, \phi^{(S)}} L_{\text{base}}(\phi^{(S)}, \pi^{(S)}) + \lambda\big(L_P(\phi^{(S)}; \hat{P}) + L_R(\phi^{(S)}; \hat{R})\big)$.
  if $t \mid m$ **then**
    Update the stable encoder $\hat{\phi}^{(S)} \leftarrow \phi^{(S)}$.

Algorithm 14: Target Task Learning with Transferred Dynamics Models

Input: Regularization weight λ; dynamics models $\hat{P}$ and $\hat{R}$ learned in the source task.
Initialize encoder $\phi^{(T)}$, policy $\pi^{(T)}$
for $t = 0, 1, \cdots$ **do**
    Take action $a_t \sim \pi^{(T)}(\phi^{(T)}(o_t^{(T)}))$, get next observation $o_{t+1}^{(T)}$ and reward r_t, store to buffer.
    Sample a mini-batch $\{o_i, a_i, r_i, o_i'\}_{i=1}^N$ from the buffer.
    Update encoder and policy by $\min_{\phi^{(T)}, \pi^{(T)}} L_{\text{base}}(\phi^{(T)}, \pi^{(T)}) + \lambda\big(L_P(\phi^{(T)}; \hat{P}) + L_R(\phi^{(T)}; \hat{R})\big)$, where L_P and L_R are defined in Equation (7.3).

7.4.2 Algorithm: Learning and Transferring Model-based Regularizer

Our goal is to use the knowledge learned in the source task to learn a good representation in the target task, such that the agent learns the target task more easily than learning from scratch. Since we focus on developing a generic transfer mechanism, the base learner can be any DRL algorithms. We use L_{base} to denote the loss function of the base learner.

As motivated in Section 7.4.1, we propose to learn policy-independent dynamics models for producing high-quality representations: (1) $\hat{P}$ which predicts the representation of the next state based on current state representation and action, and (2) $\hat{R}$ which predicts the immediate reward based on current state representation and action. For a batch of N transition samples $\{o_i, a_i, o_i', r_i\}_{i=1}^N$, define the transition loss and the reward loss as:

$$L_P(\phi, \hat{P}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{P}(\phi(o_i), a_i) - \bar{\phi}(o_i'))^2, \quad L_R(\phi, \hat{R}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{R}(\phi(o_i), a_i) - r_i)^2 \tag{7.3}$$

where $\bar{\phi}(o_i')$ denotes the representation of the next state o_i' with stop gradients. In order to fit a more diverse state distribution, transition samples are drawn from an off-policy buffer, which stores shuffled past trajectories.

The learning procedures for the source task and the target task are illustrated in Algorithm 13 and Algorithm 14, respectively. Figure 7.2 depicts the architecture of the learning model for both source and target tasks. $z = \phi(o)$ and $z' = \bar{\phi}(o')$ are the encoded observation and next observation. Given the current encoding z and the action a, the dynamics models $\hat{P}$ and $\hat{R}$ return the predicted next encoding $\hat{z}' = \hat{P}(z, a)$ and predicted reward $\hat{r} = \hat{R}(z, a)$. Then the transition loss is the mean squared error (MSE) between z' and $\hat{z}'$ in a batch; the reward loss is the MSE between r and $\hat{r}$ in a batch.

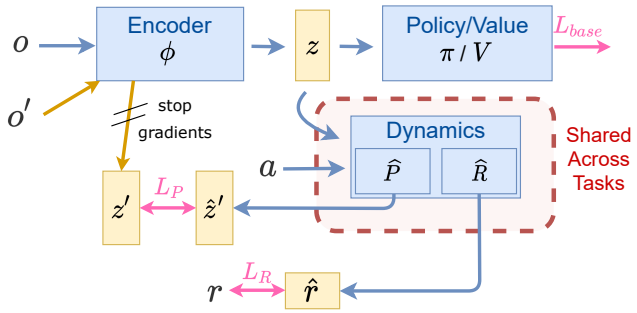


Figure 7.2: The architecture of proposed method. $\hat{P}$ and $\hat{R}$ are learned in the source task, then transferred to the target task and fixed during training.

In the source task (Algorithm 13): dynamics models $\hat{P}$ and $\hat{R}$ are learned by minimizing L_P and L_R, which are computed based on a recent copy of encoder called stable encoder $\hat{\phi}^{(S)}$ (Line 5). The computation of the stable encoder is to help the dynamics models converge, as the actual encoder $\phi^{(S)}$ changes at every step. Note that a stable copy of the network is widely used in many DRL algorithms (e.g. the target network in DQN), which can be directly regarded as $\hat{\phi}^{(S)}$ without maintaining an extra network. The actual encoder $\phi^{(S)}$ is regularized by the auxiliary dynamics models $\hat{P}$ and $\hat{R}$ (Line 6).

In the target task (Algorithm 14): dynamics model $\hat{P}$ and $\hat{R}$ are transferred from the source task and fixed during learning. Therefore, the learning of $\phi^{(T)}$ is regularized by static dynamics models, which leads to faster and more stable convergence than naively learning an auxiliary task.

Relation and Difference with Model-based RL and Bisimulation Metrics. Learning a

dynamics model is a common technique in model-based RL [196, 197], whose goal is to learn an accurate world model and use the model for planning. The dynamics model could be learned on either raw observations or representations. In our framework, we also learn a dynamics model, but the model serves as an auxiliary task, and learning is still performed by the model-free base learner with L_{base}. Bisimulation methods [180, 198] aim to approximate the bisimulation distances among states by learning dynamics models, whereas we do not explicitly measure the distance among states. Note that we also do not require a reconstruction loss that is common in literature [186].

7.4.3 Theoretical Analysis: Benefits of Transferable Dynamics Model

The algorithms introduced in Section 7.4.2 consist of two designs: learning a latent dynamics model as an auxiliary task, and transferring the dynamics model to the target task. In this section, we show theoretical justifications and practical advantages of our proposed method. We aim to answer the following two questions: *(1) How does learning an auxiliary dynamics model help with representation learning? (2) Is the auxiliary dynamics model transferable?*

For notational simplicity, let P_a and R_a denote the transition and reward functions associated with action $a \in \mathcal{A}$. Note that P_a and R_a are independent of any policy. We then define the sufficiency of a representation mapping w.r.t. dynamics models as below.

Definition 67 (Policy-independent Model Sufficiency)**.** *For an MDP $\mathcal{M}$, a representation mapping ϕ is sufficient for its dynamics $(P_a, R_a)_{a\in\mathcal{A}}$ if $\forall a \in \mathcal{A}$, there exists functions $\hat{P}_a : \mathbb{R}^d \to \mathbb{R}^d$ and $\hat{R}_a : \mathbb{R}^d \to \mathbb{R}$ such that $\forall o \in \mathcal{S}$, $\hat{P}_a(\phi(o)) = \mathbb{E}_{o'\sim P_a(o)}[\phi(o')]$, $\hat{R}_a(\phi(o)) = R_a(o)$.*

Remarks. (1) ϕ is exactly sufficient for dynamics $(P_a, R_a)_{a\in\mathcal{A}}$ when the transition function

P is deterministic. (2) If P is stochastic, but we have $\max_{o,a} \|\mathbb{E}_{o' \sim P_a(o)}[\phi(o')] - \hat{P}_a(\phi(o))\| \leq \epsilon_P$ and $\max_{o,a} |R_a(o) - \hat{R}_a(\phi(o))| \leq \epsilon_R$, then ϕ is **(ϵ_P, ϵ_R)-sufficient** for the dynamics of $\mathcal{M}$. Next we show by Proposition 68 and Theorem 69 that learning sufficiency can be achieved via ensuring model sufficiency.

Proposition 68 (Learning Sufficiency Induced by Policy-independent Model Sufficiency)**.** *Consider an MDP $\mathcal{M}$ with deterministic transition function P and reward function R. If ϕ is sufficient for $(P_a, R_a)_{a \in \mathcal{A}}$, then it is sufficient (but not necessarily linearly sufficient) for learning in $\mathcal{M}$.*

Proposition 68 shows that, if the transition is deterministic and the model errors L_P, L_R are zero, then ϕ is exactly sufficient for learning.

Error Bound with Imperfect Representation. More generally, if the transition function P is not deterministic, and model fitting is not perfect, the learned representation can still be nearly sufficient for learning as characterized by Theorem 69 below, which is extended from a variant of the value difference bound derived by [187]. Proposition 68 and Theorem 69 justify that learning the latent dynamics model as an auxiliary task encourages the representation to be sufficient for learning. The model error L_P and L_R defined in Section 7.4.2 can indicate how good the representation is.

We follow the analysis by [187] and define a new MDP under the representation mapping ϕ: $\tilde{\mathcal{M}} = \langle \tilde{\mathcal{S}}, \mathcal{A}, \tilde{P}, \tilde{R}, \gamma \rangle$, where for all $o \in \mathcal{S}$, $\phi(o) \in \tilde{\mathcal{S}}$, $\tilde{P}(\phi(o), a) = \hat{P}_a \phi(o)$, $\tilde{R}(\phi(o), a) = \hat{R}_a \phi(o)$. Let $\tilde{V}$ denote the value function in $\tilde{\mathcal{M}}$, and let $\tilde{\pi}$ denote a policy in $\tilde{\mathcal{M}}$. We make the following mild assumption

Assumption 3. *There exists a constant $K_{\phi,V}$ such that*

$$|\tilde{V}_{\tilde{\pi}}(\phi(o_1)) - \tilde{V}_{\tilde{\pi}}(\phi(o_2))| \leq K_{\phi,V} \|\phi(o_1) - \phi(o_2)\|, \forall \tilde{\pi} : \tilde{O} \to \mathcal{A}, o_1, o_2 \in \mathcal{S}.$$

This assumption is mild as any MDP with bounded reward has bounded value functions.

Theorem 69. *For an MDP $\mathcal{M}$, if representation mapping ϕ is (ϵ_P, ϵ_R)-sufficient for the dynamics of $\mathcal{M}$, then approximate policy iteration with approximation operator $\mathcal{H}_\phi$ starting from any initial policy $\pi_0 \in \Pi_\phi^D$ satisfies*

$$\limsup_{k\to\infty} \|Q^* - Q^{\pi_k}\|_\infty \leq \frac{2\gamma^2}{(1-\gamma)^3}(\epsilon_R + \gamma\epsilon_P K_{\phi,V}). \tag{7.4}$$

where $K_{\phi,V}$ is an upper bound of the value Lipschitz constant as defined in Assumption 3.

Transferring Model to Get Better Representation in Target. Although Proposition 68 shows that learning auxiliary dynamics models benefits representation learning, finding the optimal solution is non-trivial since one still has to learn $\hat{P}$ and $\hat{R}$. Therefore, the main idea of our algorithm is to transfer the dynamics models $\hat{P}, \hat{R}$ from the source task to the target task, to ease the learning in the target task. Theorem 70 below guarantees that transferring the dynamics models is feasible. Our experimental result in Section 7.5 verifies that learning with transferred and fixed dynamics models outperforms learning with randomly initialized dynamics models.

Theorem 70 (Transferable Dynamics Models)**.** *Consider a source task $\mathcal{M}^{(S)}$ and a target task $\mathcal{M}^{(T)}$ with deterministic transition functions. Suppose $\phi^{(S)}$ is sufficient for $(P_a^{(S)}, R_a^{(S)})_{a\in\mathcal{A}}$ with functions $\hat{P}_a, \hat{R}_a$, then there exists a representation $\phi^{(T)}$ satisfying $\hat{P}_a(\phi(o)) = \mathbb{E}_{o'\sim P_a^{(T)}(o)}[\phi(o')]$, $\hat{R}_a(\phi(o)) = R_a^{(T)}(o)$, for all $o \in \mathcal{S}^{(T)}$, and $\phi^{(T)}$ is sufficient for learning in $\mathcal{M}^{(T)}$.*

Theorem 70 shows that the learned latent dynamics models $\hat{P}, \hat{R}$ are transferable from the source task to the target task. For simplicity, Theorem 70 focuses on exact sufficiency as in Proposition 68, but it can be easily extended to ϵ-sufficiency if combined with Theorem 69. Proofs for Proposition 68, Theorem 69 and Theorem 70 are all provided in Section 7.7.1.

Trade-off between Approximation Complexity and Representation Complexity. As

suggested by Proposition 68, fitting policy-independent dynamics encourages the representation to be sufficient for learning, but not necessarily linearly sufficient. Therefore, we suggest using a *non-linear policy/value head* following the representation to reduce the approximation error. In contrast, linear sufficiency can be achieved if ϕ is made linearly sufficient for P_π and R_π for all $\pi \in \Pi_\phi^D$, where P_π and R_π are transition and reward functions induced by policy π. It can be found from Definition 65 that ϕ is sufficient as long as it represents Q_π for all $\pi \in \Pi_\phi^D$. Fitting various value functions to improve representation quality is proposed by some prior works [182, 183] and shown to be effective. However, learning and fitting many policy values could be computationally expensive, and is not easy to be applied to transfer learning between tasks with different observation spaces. Can we regularize the representation with the latent dynamics instead of policy values? Proposition 71 below shows that if ϕ is linearly sufficient for all dynamics pairs (P_π, R_π) induced by policies in Π_ϕ^D and the dynamics pairs associated with all actions, then ϕ is linearly sufficient for learning.

For notation simplicity, assume the state space is finite. Then, let (P_π, R_π) be the transition matrix and reward vector induced by policy π, i.e., $P_\pi[i,j] = \mathbb{E}_{a\sim\pi(o_i)}[P(o_j|o_i,a)]$, $R_\pi[i] = \mathbb{E}_{a\sim\pi(o_i)}[R(o_i,a)]$. Similarly, let $P_a[i,j] = P(o_j|o_i,a)$, $R_a[i] = R(o_i,a)$. We let Φ denote the representation matrix, where the i-th row of Φ refers to the feature of the i-the observation.

Proposition 71 (Linear Sufficiency Induced by Policy-based Model Sufficiency)**.** *For representation Φ, if for all $\pi \in \Pi_\phi^D, a \in \mathcal{A}$, there exist $\hat{P}_\pi, \hat{R}_\pi, \hat{P}_a, \hat{R}_a$ such that $\Phi\hat{P}_\pi = P_\pi\Phi$, $\Phi\hat{R}_\pi = R_\pi$, $\Phi\hat{P}_a = P_a\Phi, \Phi\hat{R}_a = R_a$, i.e. Φ is linearly sufficient both policy-based dynamics and policy-independent dynamics models, i.e., then Φ is linearly sufficient for a task $\mathcal{M}$ w.r.t. approximation operator $\mathcal{H}_\phi$.*

Proposition 71 proven in Section 7.7.1.3 suggests that we can let representation fit (P_π, R_π) for many different π's. However, it could be computationally intractable since the policy space is large. More importantly, it is not memory-friendly to store and transfer a large number of dynamics models for all $(\hat{P}_\pi, \hat{R}_\pi)$. In our Proposition 68, we show that learning sufficiency can be induced by policy-independent model sufficiency, which is much simpler as there is no need to learn and store $(\hat{P}_\pi, \hat{R}_\pi)$ for many policies. As a trade-off, the policy-independent model induces non-linear sufficiency instead of linear sufficiency, requiring a more expressive approximation operator.

However, using this method for transfer learning is expensive in terms of both computation and memory, as it requires to learn P_π and R_π for many different π's and store these models for transferring to the target task. Therefore, there is a trade-off between approximation complexity and representation complexity. Learning a linearly sufficient representation reduces the complexity of the approximation operator. But it requires more complexity in the representation itself as it has to satisfy much more constraints. To develop a practical and efficient transfer method, *we use a slightly more complex approximation operator (non-linear policy head) while keeping the auxiliary task simple and easy to transfer across tasks.*

7.5 Empirical Results

We empirically evaluate our transfer learning algorithm in various environments and multiple observation-change scenarios. Detailed experiment setup and hyperparameters are in Section 7.7.2.

7.5.1 Experiment Setting

Baselines. To verify the effectiveness of our proposed transfer learning method, we compare our transfer learning algorithm with 4 baselines: (1) *Single*: a single-task base learner. (2) *Auxiliary*: learns auxiliary models from scratch to regularize representation. (3) *Fine-tune*: loads and freezes the source policy head, and retrains an encoder in the target task. (4) *Time-aligned* [176]: supposes the target task and the source task proceed to the same latent state given the same action sequence, and pre-trains a target-task encoder with saved source-task trajectories. More details of baseline implementations are in Section 7.7.2.1.

Scenarios. As motivated in Section 7.1, there are many scenarios where one can benefit from transfer learning across observation feature spaces. We evaluate our proposed transfer algorithm in 7 environments that fit various scenarios, to simulate real-world applications:

(1) *Vec-to-pixel:* a novel and challenging scenario, where the source task has low-dimensional vector observations and the target task has pixel observations. We use 3 vector-input environments CartPole, Acrobot and Cheetah-Run as source tasks, and use the rendered image in the target task.

(2) *More-sensor:* another challenging scenario where the target task has a lot more sensors than the source task. We use 3 MuJoCo environments: HalfCheetah, Hopper and Walker2d, whose original observation dimensions are 17, 11 and 17, respectively. We add mass-based inertia and velocity (provided by MuJoCo's API), resulting in 145, 91, 145 dimensions in the corresponding target tasks.

(3) *Broken-sensor:* we use an existing game 3DBall contained in the Unity ML-Agents Toolkit [169], which has two different observation specifications that naturally fit our transfer setting: the source

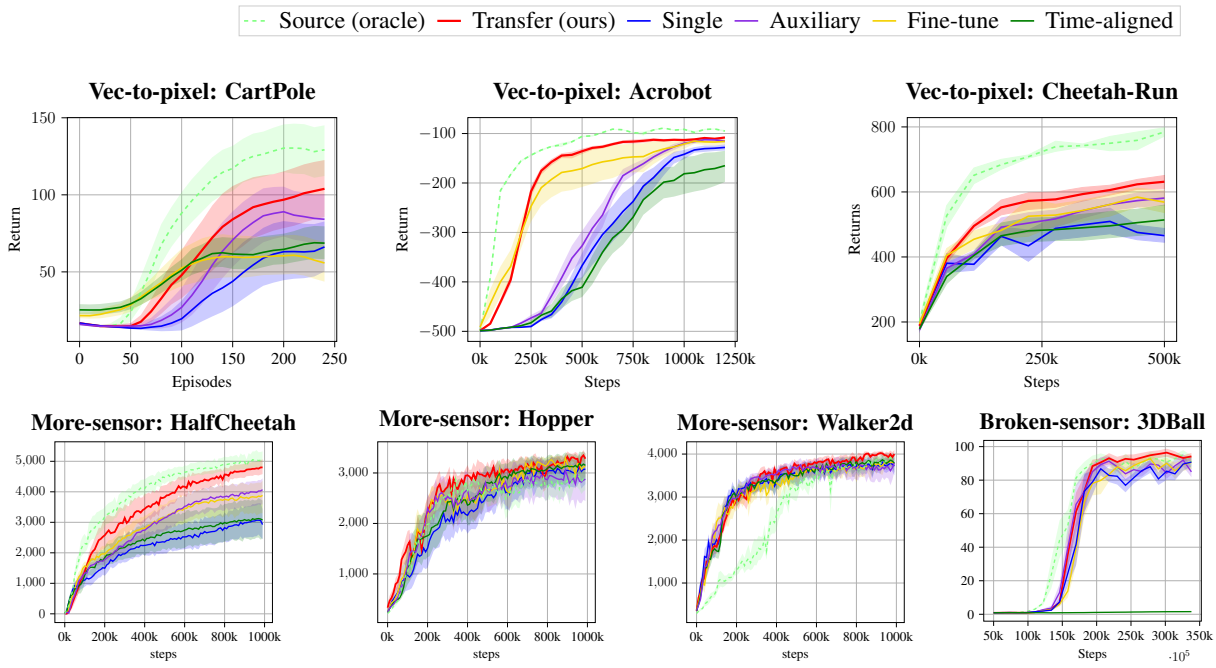


Figure 7.3: Our proposed transfer method outperforms all baselines in target tasks over all tested scenarios. (The dashed green lines are the learning curves in source tasks.) Results are averaged over 10 random seeds.

observation has 8 features containing the velocity of the ball; the target observation does not have the ball's velocity, thus the agent has to stack the past 9 frames to infer the velocity. Please see Section 7.7.2.2 for more detailed descriptions of all the 7 environments.

Base DRL Learners. What we propose is a transfer learning mechanism that can be combined with any existing DRL methods. For environments with discrete action spaces (CartPole, Acrobot), we use the DQN algorithm [1], while for environments with continuous action spaces (Cheetah-Run, HalfCheetah, Hopper, Walker2d, 3DBall), we use the SAC algorithm [98]. To ensure a fair comparison, we use the same base DRL learner with the same hyperparameter settings for all tested methods, as detailed in Section 7.7.2.3. As is common in prior works, our implementation of the RL algorithms is mostly a proof of concept, thus many advanced training techniques are not included (e.g. Rainbow DQN).

7.5.2 Experiment Results

Experimental results on all tested environments are shown in Figure 7.3. We can see that our proposed transfer method learns significantly better than the single-task learner, and also outperforms all baselines in the challenging target tasks. Our transfer method outperforms Auxiliary since it transfers dynamics model from the source task instead of learning it from scratch, and outperforms Fine-tine since it regularizes the challenging encoder learning with a model-based regularizer. The Time-aligned method, although requires additional pre-training that is not shown in the figures, does not work better than Single in most environments, because the time-based alignment assumption may not hold as discussed in Section 7.7.2.1. In some environments (e.g. Hopper, Walker2d, 3DBall), our transfer algorithm even achieves better asymptotic performance than the source-task policy, which suggests that our method can be used for improving the policy with incremental observation design. *To the best of our knowledge, we are the first to achieve effective knowledge transfer from a vector-input environment to a pixel-input environment without any pre-defined mappings.*

7.5.3 Ablation Study and Discussion

Ablation Study: Transferring Different Components. Figure 7.4 shows the ablation study of our method in continuous control tasks. We compare our method with the following variants: (1) learning auxiliary tasks without transfer, (2) only transferring transition models $\hat{P}$ and (3) only transferring reward models $\hat{R}$.

Compared with the single-task learning baseline (the blue curves), we find that all the variants of our method can make some improvements, which suggests that learning dynamics models

as auxiliary tasks, transferring $\hat{P}$ and $\hat{R}$ are all effective designs for accelerating the target task learning. Finally, our method (the red curves) that combines the above components achieves the best performance, justifying the effectiveness of our transfer algorithm.

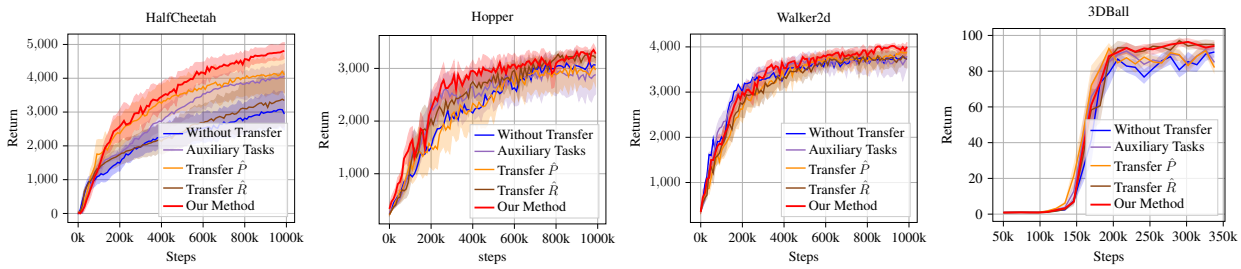


Figure 7.4: Ablation study of our method on different transferred components.

Sanity Check: Effectiveness of the Proposed Transfer Method. We conduct another ablation study to evaluate each component of our algorithm in the CartPole environment as shown in Figure 7.5. We find that when transferring the dynamics models with only a linear value head (the **green** curve), the agent fails to learn a good policy as we analyzed in Section 7.4.3. If the dynamics models $(\hat{P}, \hat{R})$ are randomly generated instead of being transferred from the source task (the **orange** curve), the agent does not learn, either. More importantly, if we learn dynamics models as auxiliary tasks in the target task without transferring them from the source (the **purple** curve), the agent learns a little better than a vanilla agent, but is worse than our proposed transfer algorithm. These empirical results have verified our theoretical insights and shown the effectiveness of our algorithm design.

Hyper-parameter Test. Figure 7.6 further visualizes how the hyperparameter λ (regularization weight) influences the transfer performance in the Vec-to-pixel CartPole environment. It can be found that the agent generally benefits from a larger λ, which suggests that the model-based regularization has a positive impact on the learning performance. For a wide range of λ's, the agent always outperforms the learner without transfer (the learner with $\lambda = 0$). Therefore,

our algorithm is not sensitive to the hyperparameter λ, and a larger λ is preferred to get better performance.

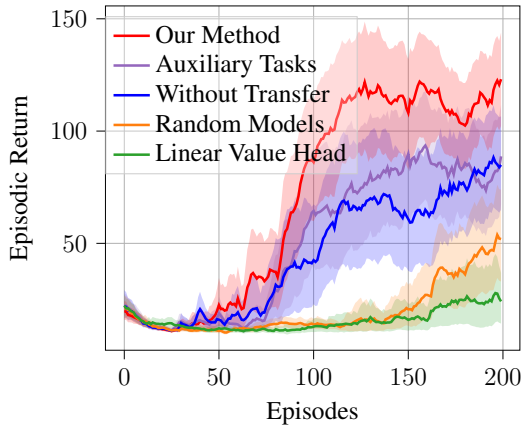


Figure 7.5: In the Vec-to-pixel CartPole environment, sanity check verifies the effectiveness of our algorithm design. Results are averaged over 20 random seeds.

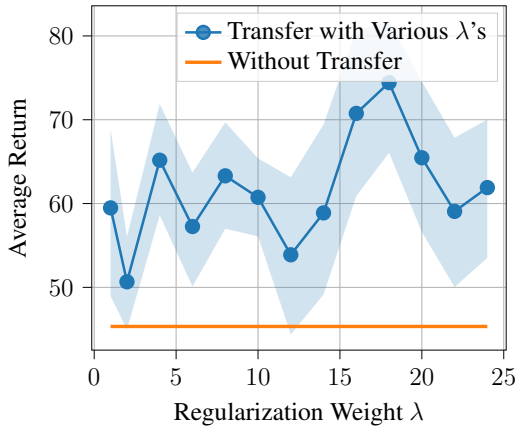


Figure 7.6: In the Vec-to-pixel CartPole environment, under different selections of hyperparameter λ, the algorithm works better than learning from scratch (when $\lambda = 0$). Results are averaged over 20 random seeds.

7.6 Conclusion

In this chapter, we identify and propose a solution to an important but rarely studied problem: transferring knowledge between tasks with drastically different observation spaces where inter-task mappings are not available. We propose to learn a latent dynamics model in the source task and transfer the model to the target task to facilitate representation learning. Theoretical analysis and empirical study justify the effectiveness of the proposed algorithm.

Potential Limitations and Solutions. As Figure 6.3 shows, in some environments such as HalfCheetah, our transfer algorithm significantly outperforms baselines without transfer. But in Walker2d, the improvement is less significant, although transferring is still better than not transferring. This phenomenon is common in model-based learning [167], as state predicting in Walker2d is harder than that in HalfCheetah due to the complexity of the dynamics. Therefore,

we suggest using our method to transfer when the learned models $(\hat{P}, \hat{R})$ in the source task are relatively good (error is low). More techniques of improving model-based learning, such as bisimulation [180, 198], can be applied to further improve the transfer performance.

7.7 Supplemental Materials: Proofs and Additional Details

7.7.1 Theoretical Analysis and Proofs

7.7.1.1 Proof of Lemma 66

Proof of Lemma 66. We first show that policy iteration with approximation operator $\mathcal{H}_\phi$ starting from a policy $\pi_0 \in \Pi_\phi^D$ generates a sequence of policies that are in Π_ϕ^D. That is, for all π_k, and any $o_1, o_2 \in \mathcal{S}$, $\pi_k(o_1) = \pi_k(o_2)$ if $\phi(o_1) = \phi(o_2)$. We prove this claim by induction.

Base case: when $k = 0$, $\pi_0 \in \Pi_\phi^D$.

Inductive step: assume $\pi_k \in \Pi_\phi^D$ for $k \geq 0$, then for iteration $k+1$, we know that

$$\pi_{k+1}(o) := \text{argmax}_a Q_k(o, a) \tag{7.5}$$

where $Q_k = \mathcal{H}_\phi Q_{\pi_k}$.

Based on the definition of $\mathcal{H}_\phi$, $Q_k(o, a) = f(\phi(o); \theta_a)$ for some θ_a. Hence, if o_1 and o_2 have the same representation, $Q_k(o_1, \cdot)$ and $Q_k(o_2, \cdot)$ are equal. As a result, $\pi_{k+1}(o_1)$ and $\pi_{k+1}(o_2)$ are equal, so $\pi_{k+1} \in \Pi_\phi^D$.

Next we prove the error bound in Lemma 66. We start by restating the error bounds for approximate policy iteration by [194]:

$$\text{limsup}_{k\to\infty} \|V^* - V^{\pi_k}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \text{sup}_k \|V_k - V^{\pi_k}\|_\infty \tag{7.6}$$

where π_k is the policy in the k-th iteration. Then we extend the above result to the action value

Q.

For any π_k during the policy iteration (as proven above, $\pi_k \in \Pi_\phi^D$), if ϕ is ϵ-sufficient for $\mathcal{M}$ as defined in Definition 65 with ℓ_∞ norm, then we have $\|Q_k - Q_{\pi_k}\|_\infty \leq \epsilon$. That is, $\forall o \in \mathcal{S}, a \in \mathcal{A}, |Q_k(o,a) - Q_{\pi_k}(o,a)| \leq \epsilon$. Therefore, $\forall o \in \mathcal{S}$,

$$|V_k(o) - V_{\pi_k}(o)| = |\sum_{a \in \mathcal{A}} \pi(a|o)(Q_k(o,a) - Q_{\pi_k}(o,a))| \leq \epsilon \tag{7.7}$$

On the other hand, we can derive

$$\|Q^* - Q_{\pi_k}\|_\infty = \max_{o,a} |Q^*(o,a) - Q_{\pi_k}(o,a)| \tag{7.8}$$

$$= \max_{o,a} |R(o,a) + \gamma \sum_{o' \in \mathcal{S}} P(o'|o,a)V^*(o') - R(o,a) - \gamma \sum_{o' \in \mathcal{S}} P(o'|o,a)V_{\pi_k}(o')| \tag{7.9}$$

$$= \gamma \|V^* - V_{\pi_k}\|_\infty \tag{7.10}$$

Combining Equation (7.7) and (7.10), we obtain

$$\text{limsup}_{k \to \infty} \|Q^* - Q_{\pi_k}\|_\infty \leq \frac{2\gamma^2}{(1-\gamma)^2}\epsilon. \tag{7.11}$$

□

7.7.1.2 Proof of Proposition 68

Proof of Proposition 68. Given that P is deterministic for $o \in \mathcal{S}, a \in \mathcal{A}$, we slightly abuse notation and let $o' = P(o,a) = P_a(o)$ if $P(o'|o,a) = 1$. If ϕ is sufficient for the dynamics models, i.e. $\forall o \in \mathcal{S}, a \in \mathcal{A}$, $\hat{P}_a\phi(o) = \phi(P_a(o))$, $\hat{R}_a\phi(o) = R_a(o)$. Then, we can define a new MDP $\tilde{\mathcal{M}} = \langle \tilde{\mathcal{S}}, \mathcal{A}, \tilde{P}, \tilde{R}, \gamma \rangle$, where for all $o \in \mathcal{S}$, $\phi(o) \in \tilde{\mathcal{S}}$, and $\tilde{P}(\phi(o), a) = \hat{P}_a\phi(o) = \phi(P(o,a))$, $\tilde{R}(\phi(o), a) = \hat{R}_a\phi(o) = R(o,a)$.

Any policy $\pi \in \Pi_\phi^D$, based on the definition of Π_ϕ^D, can be written as $\tilde{\pi} \circ \phi$, where $\tilde{\pi}$ is a

deterministic policy in $\tilde{\mathcal{M}}$. Next, we show that for all $o \in \mathcal{S}, a \in \mathcal{A}$, $Q_\pi(o) = \tilde{Q}_{\tilde{\pi}}(\phi(o))$.

By definition of the Q value, we know

$$Q_\pi(o, a) = \mathbb{E}_{\pi,P}[\sum_{t=0}^{\infty} \gamma^t R(o_t, a_t) | o_0 = o, a_0 = a] \tag{7.12}$$

$$\tilde{Q}_{\tilde{\pi}}(\phi(o)) = \mathbb{E}_{\tilde{\pi},\tilde{P}}[\sum_{t=0}^{\infty} \gamma^t \tilde{R}(\tilde{o}_t, \tilde{a}_t) | \tilde{o}_0 = \phi(o), \tilde{a}_0 = a] \tag{7.13}$$

We claim that in the above equations, $\tilde{o}_t = \phi(o_t)$, $\tilde{a}_t = a_t$, for all $t \geq 0$. We prove the claim by induction.

When $t = 0$, the claim holds as $\tilde{o_0} = \phi(o_0) = \phi(o)$, $\tilde{a_0} = a_0 = a$.

Then, with inductive hypothesis that $\tilde{o_t} = \phi(o_t)$, $\tilde{a_t} = a_t$, we show the claim holds for $t+1$:

Action: $a_{t+1} = \pi(o_{t+1}) = \tilde{\pi}(\phi(o_{t+1})) = \tilde{a}_{t+1}$.

State: $\tilde{o}_{t+1} = \tilde{P}(\tilde{o_t}, a_t) = \tilde{P}(\phi(o_t), a_t) = \phi(P(o_t, a_t)) = \phi(o_{t+1})$.

Hence, we have shown $\tilde{o}_t = \phi(o_t)$, $\tilde{a}_t = a_t$, for all $t \geq 0$, then the reward in the t-th step of Equation (7.12) and (7.13) are the same, as $\tilde{R}(\tilde{o_t}, \tilde{a_t}) = \tilde{R}(\phi(o_t), a_t) = R(o_t, a_t)$. Therefore, $Q_\pi(o) = \tilde{Q}_{\tilde{\pi}}(\phi(o))$.

Therefore, for any $\pi \in \Pi_\phi^D$, its action value can be represented by $\tilde{Q}_{\tilde{\pi}} \circ \phi$. Therefore, ϕ is sufficient for learning in $\mathcal{M}$.

Next, we show that ϕ **is not necessarily linearly sufficient** for learning the task.

Consider an arbitrary policy $\pi \in \Pi_\phi^D$. Without loss of generality, suppose R_π is linearly

represented by ϕ, i.e. $R_a(o,a) = \phi(o)^\top \hat{R}_a$, then we have

$$\begin{aligned} R_\pi(o) &= \sum_{a\in\mathcal{A}} \pi(a|\phi(o)) R_a(o,a) \\ &= \sum_{a\in\mathcal{A}} \pi(a|\phi(o)) \phi(o)^\top \hat{R}_a \\ &= \langle \pi(\phi(o)), \hat{R}^\top \phi(o) \rangle \\ &= \langle \hat{R}\pi(\phi(o)), \phi(o) \rangle \end{aligned}$$

where $\hat{R} := [\hat{R}_{a_1}; \hat{R}_{a_1}; \cdots; \hat{R}_{a_{|\mathcal{A}|}}]$. We can find that unless π always takes the same action for all input states, ϕ is not guaranteed to linearly encode R_π.

Similarly, for P_π, suppose $P_a(\cdot|o,a) = \phi(o)^\top \hat{P}_a$ we have

$$\begin{aligned} \phi(P_\pi(o)) &= \sum_{a\in\mathcal{A}} \pi(a|\phi(o)) P_a(\cdot|o,a) \\ &= \sum_{a\in\mathcal{A}} \pi(a|\phi(o)) \phi(o)^\top \hat{P}_a \\ &= \hat{P}(\pi(\phi(o)), \phi(o), I) \end{aligned}$$

where $\hat{P} := [\hat{P}_{a_1}; \hat{P}_{a_1}; \cdots; \hat{P}_{a_{|\mathcal{A}|}}]$ is an $|\mathcal{A}| \times d \times d$ tensor, and $\hat{P}(\cdot,\cdot,\cdot)$ denotes the multi-linear operation. Hence, if π takes different actions in different states, ϕ may not linearly encode the transition, either.

Therefore, ϕ is not guaranteed to linearly encode R_π and P_π, and thus is not guaranteed to linearly encode V_π and Q_π.

□

7.7.1.3 Proof of Proposition 71

Proof of Proposition 71. We first show that for any $\pi \in \Pi$, if ϕ is linearly sufficient for (P_π, R_π), then there exists a vector $\omega \in \mathbb{R}^k$ such that $V_\pi = \hat{V}_\pi = \Phi\omega$.

Since Φ is linearly sufficient for P_π and R_π, we have $\Phi\hat{P}_\pi = P_\pi\Phi$ and $\Phi\hat{R}_\pi = R_\pi$ for some $\hat{P}_\pi$ and $\hat{R}_\pi$. Let $\omega = (I - \gamma\hat{P}_\pi)^\dagger\hat{R}_\pi$, then the Bellman error of $\hat{V}_\pi = \Phi\omega$ can be computed as

$$\begin{aligned} R_\pi + \gamma P_\pi\hat{V}_\pi - \hat{V}_\pi &= R_\pi + \gamma P_\pi\Phi\omega - \Phi\omega \\ &= \Phi\hat{R}_\pi + \gamma\Phi\hat{P}_\pi\omega - \Phi\omega \\ &= \Phi(\hat{R}_\pi - (I - \gamma\hat{P}_\pi)(I - \gamma\hat{P}_\pi)^\dagger\hat{R}_\pi) \\ &= \Phi(\hat{R}_\pi - \hat{R}_\pi) \\ &= 0 \end{aligned}$$

Therefore, $\hat{V}_\pi$ is a fixed point of the Bellman operator $\mathcal{T}^\pi$, and thus equal to V_π.

Next, as we know that $Q_\pi(\cdot, a) = R_a + \gamma\langle P_a, V_\pi\rangle$, and $\Phi\hat{P}_a = P_a\Phi, \Phi\hat{R}_a = R_a$, we can obtain

$$\begin{aligned} Q_\pi(\cdot, a) &= R_a + \gamma\langle P_a, V_\pi\rangle \\ &= \Phi\hat{R}_a + \gamma P_a\Phi\omega \\ &= \Phi\hat{R}_a + \gamma\Phi\hat{P}_a\omega \\ &= \Phi(\hat{R}_a + \gamma\hat{P}_a\omega) \end{aligned}$$

Therefore, for any $\pi \in \Pi_\phi^D$, Q_π can be linearly represented by Φ, and thus Φ is linearly sufficient for learning by definition.

□

7.7.1.4 Proof of Theorem 69

Proof of Theorem 69. Lemma 2 in [187] is based on one policy in the induced MDP and bounded model errors. We can replace the Wasserstein distance $\mathcal{W}(\phi P(\cdot|o,a), \tilde{P}(\cdot|\phi(o),a))$ by the Euclidean distance $\|\phi P(o,a), \tilde{P}(\phi(o),a)\|$ as we focus on deterministic transitions.

For any policy $\pi \in \Pi_\phi^D$ that can be written as $\tilde{\pi} \circ \phi$, we have

$$|Q_\pi(o,a) - \tilde{Q}_{\tilde{\pi}}(\phi(o),a)| \leq \frac{\epsilon_R + \gamma K_{\phi,V} \epsilon_P}{1-\gamma} \tag{7.14}$$

Therefore, ϕ is $(1-\gamma)^{-1}(\epsilon_R + \gamma K_{\phi,V}\epsilon_P)$-sufficient for learning $\mathcal{M}$.

Combined with Lemma 66, we can obtain the bound in Theorem 69.

□

7.7.1.5 Proof of Theorem 70

Proof of Theorem 70. First of all, if there exists $\phi^{(T)}$ satisfying $\hat{P}_a(\phi(o_i)) = P_a^{(T)}[i]\Phi^{(T)}$, $\hat{R}_a(\phi(o_i)) = R_a^{(T)}[i]$, $\forall o_i \in \mathcal{S}^{(T)}$, then it is sufficient for the dynamics of the target task, and thus sufficient for learning the target task as stated in Proposition 68. Therefore, our focus is to show the existence of such a representation.

As $\phi^{(S)}$ is sufficient for $P_a^{(S)}$ and $R_a^{(S)}$ for all $a \in \mathcal{A}$, we have

$$\hat{P}_a(\phi^{(S)}(o^{(S)})) = P^{(S)}(o^{(S)},a)\Phi^{(S)} = \phi^{(S)}(P^{(S)}(o^{(S)},a)) \tag{7.15}$$

$$\hat{R}_a(\phi^{(S)}(o^{(S)})) = R^{(S)}(o^{(S)},a) \tag{7.16}$$

where we let $P^{(S)}(o^{(S)},a)$ denote the next state of $(o^{(S)},a)$, given that P is deterministic.

Based on Assumption 2, we know that there exists a function f such that $\forall o^{(T)} \in \mathcal{S}^{(T)}$, $f(o^{(T)}) \in \mathcal{S}^{(S)}$, and $f(P^{(T)}(o^{(T)},a)) = P^{(S)}(f(o^{(T)}),a)$, $R^{(T)}(o^{(T)},a) = R^{(S)}(f(o^{(T)}),a)$. Hence, we can

obtain

$$\hat{P}_a(\phi^{(S)}(f(o^{(T)}))) = \phi^{(S)}(P^{(S)}(f(o^{(T)}), a)) = \phi^{(S)}(f(P^{(T)}(o^{(T)}, a))) \tag{7.17}$$

$$\hat{R}_a(\phi^{(S)}(f(o^{(T)}))) = R^{(S)}(f(o^{(T)}), a) = R^{(T)}(o^{(T)}, a) \tag{7.18}$$

Let $\hat{\phi}^{(T)} := \phi^{(S)} \circ f$, then we get

$$\hat{P}_a(\hat{\phi}^{(T)}(o^{(T)})) = \hat{\phi}^{(T)}(P^{(T)}(o^{(T)}, a)) \tag{7.19}$$

$$\hat{R}_a(\hat{\phi}^{(T)}(o^{(T)})) = R^{(T)}(o^{(T)}, a) \tag{7.20}$$

Therefore, $\hat{\phi}^{(T)}$ is a feasible solution satisfying model sufficiency in the target task, and thus is sufficient for learning.

Theorem 70 holds since we have shown (1) all feasible solutions to $\Phi^{(T)}\hat{P}_a = P_a^{(T)}\Phi$ and $\Phi^{(T)}\hat{R}_a = R_a^{(T)}$ are sufficient for learning in $\mathcal{M}^{(T)}$, and (2) there exists at least one feasible solution to $\Phi^{(T)}\hat{P}_a = P_a^{(T)}\Phi$ and $\Phi^{(T)}\hat{R}_a = R_a^{(T)}$.

□

7.7.2 Experiment Setting Details

7.7.2.1 Baselines

- **Single**: A DQN or SAC learner on the target domain without any auxiliary tasks.

- **Auxiliary**: On the target domain, the encoder $\phi^{(T)}$ is optimized based on the loss $L_{\text{base}}(\phi^{(T)}, \pi^{(T)}) + \lambda\Big[L_P(\phi^{(T)}; \hat{P}^{(T)}) + L_R(\phi^{(T)}; \hat{R}^{(T)})\Big]$. Compared with our transfer algorithms which transfer the learned dynamics from source domain to the target domain, it learns the dynamics model $(\hat{P}^{(T)}, \hat{R}^{(T)})$ on the target domain from scratch. Here we set λ to be the same as our transferred algorithm (values of λ are provided in Section 7.7.2.3). The purpose of this baseline

is to test whether the efficiency of our proposed transfer algorithms come from the transferred latent dynamics or from the auxiliary loss (or potentially both).

- **Fine-tune**: To test whether our transfer algorithms benefit from loading the learned policy head $\pi^{(S)}$, on the target domain, we load the weights of $\pi^{(T)}$ from the trained source policy head $\pi^{(S)}$ and train the DQN or SAC agent without any auxiliary loss.

- **Time-aligned**: Gupta et al. [176] propose to learn aligned representations for two tasks, under the assumption that the source-task agent and the target-task agent reach similar latent sates at the same time step, i.e. $\phi^{(T)}(s_t^{(T)}) = \phi^{(S)}(s_t^{(S)})$. Note that this assumption is valid when the initial state is fixed and the transitions are all deterministic. Although in our setting, the agent can not learn both tasks simultaneously, we can adapt the idea of time-based alignment and encourage the target encoder to map target observations to the source representations happening at the same time step.

  In our experiments, we store N source trajectories $\left\{s_0^i, a_0^i, s_1^i, a_1^i, ..., \right\}_{i=1}^{N}$ collected during source task training. Then on the target domain, we first collect N trajectories following the same action as the one collected from the source domain. In other words, at time step t of the i-th trajectory, we take action a_t^i. After the target trajectories are collected, we minimize the alignment loss $L_{\text{align}}(\phi^{(T)}) = \mathbb{E}\left[\left(\phi^{(T)}(s_t^{(T)}) - \phi^{(S)}(s_t^{(S)})\right)^2\right]$ to enforce that observations from source and target domain at the same time-step have the same representations.

  In our experiments, we set N to be 10% of the training trajectories. (We also experimented with larger N', for example using all the training trajectories, but the differences are minor.) In terms the alignment loss, we optimize the loss for 1000 epochs with batch size equal to 256, where at each epoch we sample a batch of paired source and target observations and

compute the alignment loss. After pre-training the target encoder, we load the weight into $\phi^{(T)}$ and resumes the normal DQN or SAC training.

Our experimental results show that, although more training steps are given to the time-aligned learner, it does not outperform the single-task learner, and sometimes fails to learn (e.g. in 3DBall). The main reason is that the time-based assumption does not hold in practice as initial states are usually randomly generated. Therefore, even though the agent exactly imitates the source-task policy at every step, the observations from source and target task do not necessarily match at every time-step. In environments with non-deterministic transitions, the state mismatch will be a more severe issue and may lead to an unreasonable encoder.

7.7.2.2 Environments

Environment Settings in Vec-to-pixel Tasks

- CartPole: The source task is the same as the ordinary CartPole environment on Gym. For the pixel-input target task, we extract the screen of the environment which is of size (400,600), and crop the pixel input to let the image be centered at the cart. The resulting observation has size (40,90) after cropping. We take the difference between two consecutive frames as the agent's observation.

- Acrobot: The source task is the same as the ordinary Acrobot environment on Gym. For the pixel-input target task, we first extract the screen of the environment which is of size (150,150), and then down-sample the image to (40,40). We also take the difference between two consecutive frames as the agent's observation.

- Cheetah-Run: The source task is the Cheetah Run Task provided by DeepMind Control Suite (DMC) [199]. For the target task, we use the image of size (84,84) rendered from the environment as the agent's observation.

Environment Settings in More-sensor Tasks For the target task of MuJoCo environments, we add the center of the mass based inertia and velocity into the observations of the agent, concatenating them with the original observation on the source task. Consequently, in the target environments, the dimensionality of the observation space on target task become much larger than that of the source task. On Hopper, the dimensionality of the target observation is 91, whereas the the source observation space only has 11 dimensions. The dimensionalities of target tasks on HalfCheetah, Hopper and Walker are 145, 91, 145 respectively.

Environment Settings in Broken-sensor Tasks 3DBall is an example environment provided by the ML-Agents Toolkit [169]. In this task, the agent (a cube) is supposed to balance a ball on its top. At every step, the agent will be rewarded if the ball is still on its top. If the ball falls off, the episode will immediately end. The highest episodic return in this task is 100. There are two versions of this game, which only differ by their observation spaces. The simpler version (named 3DBall in the toolkit) has 8 observation features corresponding to the rotation of the agent cube, and the position and velocity of the ball. The harder version (named 3DBallHard in the toolkit) does not have access to the ball velocity, but observes a stack of 9 past frames, each of which corresponds to the rotation of the agent cube, and the position of the ball, resulting in 45 observation dimensions at every step. We regard 3DBall as the source task and 3DBallHard as the target task in our experiments.

7.7.2.3 Implementation of Base DRL Algorithms and Hyper-parameter Settings

Implementation of DQN To ensure that the base learning algorithm learns the pixel-input target tasks well, we follow the existing online codebases for pre-processing, architectures and hyperparameter settings in pixel CartPole[1] and pixel Acrobot [2]. On source domain, the DQN network has a 2-layer encoder and a 2-layer Q head of hidden size 64, and the representation dimension is set as 16. For pixel-input, the encoder has three convolution layers followed by a linear layer. The number of channels of the convolutional layers are equal to 16, 32, 32, respectively (kernel size=5 for all three layers). We use the Adam optimizer with learning rate 0.001 and $\beta_1, \beta_2 = 0.9, 0.999$. The target Q network is updated every 10 iterations. In CartPole, we use a replay buffer with size 10000. In the more challenging Acrobot, we use a prioritized replay buffer with size 100000.

Implementation of SAC For MuJoCo environments, we follow an elegant open-sourced SAC implementation[3]. The number of hidden units for all neural networks is 256. The actor has a two-layer encoder and a two-layer policy head. The two Q networks both have three linear layers. The activation function is ReLU and the learning rate is $3 \cdot 10^{-4}$. We train the dynamics model and the reward model every 50k interactive steps in the source task. For the DMC environment Cheetah-Run, we follow the open-sourced SAC implementation with an autoencoder [4]. The pixel encoder has three convolution layers and one linear layer. The number of channels for all convolutional layers is 32 and the kernel size is 3. For the 3DBall environment, as it can only be

[1] https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
[2] https://github.comeyalbd2Deep_RL_Course
[3] https://github.com/pranz24/pytorch-soft-actor-critic
[4] https://github.com/denisyarats/pytorch_sac_ae

learned within the ML-Agents toolkit, we directly use the SAC implementation provided by the toolkit with the default hyperparameter settings.

Implementation of Latent Dynamics Model Note that our goal is to learn a good representation by enforcing it predicting the latent dynamics, different from model-based RL [200] that aims to learn accurate models for planning. Therefore, we let the dynamics models $\hat{P}$ and $\hat{R}$ be simple linear networks, so that the representation can be more informative in terms of representing dynamics and learning values/policies. For environments with discrete action spaces, we learn $|\mathcal{A}|$ linear transition networks and $|\mathcal{A}|$ linear reward models. For environments with continuous action spaces, we first learn an action encoder $\psi : \mathcal{A} \to \mathbb{R}^d$ with the same encoding size d as the state representation. Then, we learn a linear transition network and a linear reward network with $\hat{P}(\phi(o) \circ \psi(a))$ being the predicted next representation, and $\hat{R}(\phi(o) \circ \psi(a))$ being the predicted reward, where $\circ$ denotes element-wise product. In practice, we find this implementation achieves good performance across many environments.

In addition, note that due to the significant difference between source observation and target observation, the initial encoding scale could be very different in source and target tasks, making it hard for them to be regularized by the same dynamics model. Therefore, we normalize the output of both encoders to be a unit vector (l2 norm is 1), which remedies the potential mismatch in their scales.

Hyperparameter Settings for Transfer Learning In experiments, we find that it is better to set λ relatively large when the environment dynamics are simple and the dynamics model is of high quality. When the environment dynamics is complex, we choose to be more conservative and set λ to be smaller. Concretely, in CartPole, λ is set as 18; in 3DBall, λ is set as 10; in Acrobot, λ is set as 5; in the remaining MuJoCo environments where dynamics are more com-

plicated, λ is set as 1. Although we use different λ's in different environments based on domain knowledge, we find that different values of λ's do not have much influence on the learning performance. Figure 2.13 in Section 7.5 shows a test on the hyper-parameter λ, where we can see that our algorithm effectively transfers knowledge under various values of λ.

Regarding the representation dimension, we set it to be smaller for simpler tasks, and larger for more complex tasks. In 3DBall, we set the encoding size to be 8; in CartPole, we set the encoding size as 16; in Acrobot, we set the encoding size as 32; in Cheetah-Run, we set the encoding size as 50; in MuJoCo tasks, we set the encoding size as 256. Again, we find that the feature size does not influence the performance too much. But based on the theoretical insights of learning minimal sufficient representation [188], we believe that it is generally better to have a lower-dimensional representation while making sure it is sufficient for learning.

Chapter 8: Adapting to Various Tasks with A Single Foundation Model

8.1 Introduction

Self-supervised pretraining has been successful in a wide range of language and vision problems. Examples include BERT [201], GPT [202], MoCo [203], and CLIP [204]. These works demonstrate that one single pretrained model can be easily finetuned to perform many downstream tasks, resulting in a simple, effective, and data-efficient paradigm. When it comes to sequential decision making, however, it is not clear yet whether the successes of pretraining approaches can be easily replicated.

There are research efforts that investigate application of pretrained vision models to facilitate control tasks [205, 206]. However, there are challenges unique to sequential decision making and beyond the considerations of existing vision and language pretraining. We highlight these challenges below: (1) *Data distribution shift:* Training data for decision making tasks is usually composed of trajectories generated under some specific behavior policies. As a result, data distributions during pretraining, downstream task finetuning and even during deployment can be drastically different, resulting in a suboptimal performance [207]. (2) *Large discrepancy between tasks:* In contrast to language and vision where the underlying semantic information is often shared across tasks, decision making tasks span a large variety of task-specific configurations, transition functions, rewards, as well as action and state spaces. Consequently, it is hard to

obtain a generic representation for multiple decision making tasks. (3) *Long-term reward maximization:* The general goal of sequential decision making is to learn a policy that maximizes long-term reward. Thus, a good representation for downstream policy learning should capture information relevant for both immediate and long-term planning, which is usually hard in tasks with long horizons, partial observability and continuous control. (4) *Lack of supervision and high-quality data:* Success in representation learning often depends on the availability of high quality expert demonstrations and ground-truth rewards [191, 208]. However, for most real-world sequential decision making tasks, high-quality data and/or supervisory signals are either non-existent or prohibitively expensive to obtain.

Under these challenges, we strive for pretrained representations for control tasks that are (1) **Versatile** so as to handle a wide variety of downstream control tasks and variable downstream learning methods such as imitation and reinforcement learning (IL, RL) etc, (2) **Generalizable** to unseen tasks and domains spanning multiple rewards and agent dynamics, and (3) **Resilient** and robust to varying-quality pretraining data without supervision.

We propose a general pretraining framework named *Self-supervised Multi-task pretrAining with contRol Transformer (SMART)*, which aims to satisfy the above listed properties. We introduce *Control Transformer (CT)* which models state-action interactions from high-dimensional observations through causal attention mechanism. Different from the recent transformer-based models for sequential decision making [209] which directly learn reward-based policies, CT is designed to learn reward-agnostic representations, which enables it as a unified model to fit different learning methods (e.g. IL and RL) and various tasks. Built upon CT, we propose a control-centric pretraining objective that consists of three terms: forward dynamics prediction, inverse dynamics prediction and random masked hindsight control. These terms focus on policy-

independent transition probabilities, and encourage CT to capture dynamics information of both short-term and long-term temporal granularities. In contrast with prior pretrained vision models [205, 210] that primarily focus on learning object-centric semantics, SMART captures the essential control-relevant information which is empirically shown to be more suitable for interactive decision making. SMART produces superior performance than training from scratch and state-of-the-art (SOTA) pretraining approaches on a large variety of tasks under both IL and RL.

Summary of Contributions.

(1) We propose SMART, a generic pretraining framework for multi-task sequential decision making.

(2) We introduce the Control Transformer model and a control-centric pretraining objective to learn representation from offline interaction data, capturing both perceptual and dynamics information with multiple temporal granularities.

(3) We conduct extensive experiments on DeepMind Control Suite [199]. By evaluating SMART on a large variety of tasks under both IL and RL regimes, SMART demonstrates its *versatile* usages for downstream applications. When adapting to unseen tasks and unseen domains, SMART shows superior *generalizability*. SMART can even produce compelling results when pretrained on low-quality data that is randomly collected, validating its *resilience* property.

8.2 Related Work

Offline Pretraining of Representation for Control. Many recent works investigate pretraining representations and finetuning policies for the same task. Yang et al. [211] investigate several pretraining objectives on MuJoCo with vector state inputs. They find that many existing

representation learning objectives fail to improve the downstream task, while contrastive self-prediction obtains the best results among all tested methods. Schwarzer et al. [212] pretrain a convolutional encoder with a combination of several self-supervised objectives, achieving superior performance on the Atari 100K. However, these works just demonstrated the single-task pretraining scenario, it is not clear yet whether the methods can be extended to multi-task control. Stooke et al. [191] propose ATC, a contrastive learning method with temporal augmentation. By pretraining an encoder on expert demonstrations from one or multiple tasks, ATC outperforms prior unsupervised representation learning methods in downstream online RL tasks, even in tasks unseen during pretraining.

Pretrained Visual Representations for Control Tasks. Recent studies reveals that visual representations pretrained on control-free datasets can be transferred to control tasks. Shah et al. [213] show that that a ResNet encoder pretrained on ImageNet is effective for learning manipulation tasks. Some recent papers also show that encoders pretrained with control-free datasets can generalize well to RL settings [205, 214, 215]. However, the generalizability of the visual encoder can be task-dependent. Kadavath et al. [216] point out that ResNet pretrained on ImageNet does not help in DMC [217] environments.

Unsupervised RL. Unsupervised RL (URL) focuses on learning exploration policies [218, 219], goal-conditioned policies [220, 221] or diverse skills [222] in a task without external rewards, and finetuning the policy later when reward is specified. The unsupervised learning phase of URL is usually interactive and prolonged. Our goal, in contrast, is to train representations of states and actions from fixed offline datasets, with a focus on capturing essential and important information from raw inputs.

Sequential Decision Making with Transformers. There is a growing body of work that

uses Transformer [223] architectures to model and learn sequential decision making problems. Chen et al. [209] propose Decision Transformer (DT) for offline RL, which takes a sequence of returns, observations and actions, and outputs action predictions. Trajectory Transformer [224] also models the trajectory as a sequence of states, actions and rewards, while discretizing each dimension of state/actions. Bonatti et al. [225] propose a pretraining scheme for state-action representations in navigation scenarios using a causal transformer, which can then be finetuned with imitation learning towards different tasks for the same robot. Furuta et al. [226] propose Generalized DT that unifies a family of algorithms for future information matching with transformers. Zheng et al. [227] extend DT to online learning by blending offline pretraining and online finetuning. Transformers can also be used as world models for model-based RL [228, 229]. Recent studies show that transformer-based models can be scaled up with diverse multi-task datasets to produce generalist agents [208, 230]. Our proposed method has a similar structure that regards RL trajectories as sequential inputs. However, differently from most existing transformer models that learn policy from returns, our SMART focuses on learning control-relevant representations with self-supervised pretraining.

8.3 Background and Problem Setup

Partially Observable Markov Decision Process. We model control tasks and environments as a partially observable Markov decision process (POMDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, E \rangle$, which is a generalization of Markov decision process (MDP). Here, $\mathcal{S}$ is the underlying state space, $\mathcal{A}$ is the action space, $\mathcal{O}$ is the observation space, P is the transition kernal, R is the reward function, and E is the observation emission function with $E(o|s)$ being the probability

of observing o given state s. In practice, the observation space can be high dimensional. For example, for a mobile robot navigating with camera sensors, the images are observations and its odometry (location, orientation, and associated velocities) and the ground-truth obstacle locations form the underlying state.

Learning History-dependent Policy. At every step t, the agent receives an observation o_t based on the underlying state s_t (hidden from the agent), takes action a_t, and obtains a reward r_t and the environment proceeds to the next state s_{t+1}. Given a history of observation-action pairs of length L and the current observation, $h_t = (o_{t-L}, a_{t-L}, o_{t-L+1}, a_{t-L+1}, \cdots, o_t)$, the agent executes action a_t according to policy π: $a_t = \pi(h_t)$. The agent's goal is to learn an optimal policy π^* that maximizes the agent's cumulative reward $\mathbb{E}_P[\sum_{t=1}^{\infty} \gamma^t r_t]$.

Multi-Task Control with Shared Representation. We consider a set of multiple tasks $\mathcal{T}$ with the same dimensionality in observation space. In this work we select $\mathcal{T}$ from different environment in DeepMind Control Suite (DMC) [199], in which the agent observes an RGB image of the current state. Tasks in $\mathcal{T}$ can have entirely different state spaces $\mathcal{S}$, different action spaces $\mathcal{A}$ and different environment dynamics (P, R, E). We also define the concept of *domain* to differentiate tasks that have different state/action spaces. For example, in DMC, "hopper" and "walker" belong to different domains because they posses distinct action spaces, while "walker-walk" and "walker-run" are different tasks within the same domain. In this chapter we use the term *multi-task* to refer tasks spanning potentially multiple domains.

Pretraining-Finetuning Pipeline. Although pretraining methods are widely applied in many areas, it is not yet clear what role pretraining should play in sequential decision making tasks, especially when considering the multi-task setup. In this work, we follow ideas established in vision and language community to explicitly define our pretraining and finetuning pipeline,

Table 8.1: A comparison between pretraining and finetuning.

Pretraining phase	**Finetuning phase**
Learn generic representation	Learn policy
Offline	Offline or online
Multiple tasks	One task, seen or unseen
Reward or expert demonstration may be absent	Has reward supervision or expert demonstration
More samples	Fewer samples

which we summarize in Table 8.1. Specifically, during the pretraining phase we train representations with a possibly large offline dataset collected from a set of training tasks $\mathcal{T}_{\text{pre}} = \{\mathcal{M}_i\}_{i=1}^n$. Then, given a specific downstream task $\mathcal{M}$ which may or may not be contained in $\mathcal{T}_{\text{pre}}$, we attach a simple policy head on top of the pretrained representation[1] and train it with IL or with RL. The central tenet of pretraining is to learn generic representations which allow downstream task finetuning to be simple, effective and efficient, even under low-data regimes.

This pretraining-finetuning pipeline is a general extension of many prior settings of pretraining for decision making. For example, Stooke et al. [191] pretrain an encoder on one or multiple tasks, then learn an RL policy in downstream tasks. Their pretraining dataset is composed of expert demonstration, and the finetuning process focuses on online RL. In addition, Schwarzer et al. [212] learn representations with offline datasets, but perform pretraining and finetuning within the same task.

8.4 Proposed Approach

We propose Self-supervised Multi-task pretrAining with contRol Transformer (SMART), a general pretraining approach for multi-task sequential decision making. We first give an overview of the proposed Control Transformer (CT) and illustrate how it fits in our pretraining-finetuning

[1]The pretrained encoder can be either frozen or finetuned with the policy, depending on the task.

pipeline in Section 8.4.1. Then, we introduce our control-centric pretraining objective in Section 8.4.2.

8.4.1 Approach Overview and Model Architecture

Model Architecture of Control Transformer. Inspired by the recent success of transformer models in sequential modeling [209, 224], we propose a Control Transformer (CT). The input to the model is a control sequence of length $2L$ composed of observations and actions: $(o_t, a_t, o_{t+1}, a_{t+1}, \cdots, o_{t+L}, a_{t+L})$. Different from Decision Transformer (DT) [209], we purposefully do not include the reward signal in the control sequence to keep our representations reward-agnostic, as explained in the end of Section 8.4.1. Each element of the sequence is embedded into a d-dimensional token, with a modality-specific tokenizer jointly trained with Transformer blocks. We also learn an additional positional embedding and sum it with each token. The outputs of CT correspond to token embeddings representing each observation and action, and are represented by $\phi(o_t)$ and $\phi(a_t)$, respectively.[2] Figure 8.1 depicts the CT architecture.

Pretraining of SMART. We generate an offline dataset for pretraining which contains control trajectories generated by some behavior policies for a set of diverse tasks $\mathcal{T}_{\text{pre}}$ spanning multiple domains. During pretraining, we append several prediction heads to the transformer output, and train the entire model by minimizing the control-centric pretraining objective introduced in Section 8.4.2. These prediction heads are used to learn desired representations, will be dropped in finetuning.

Downstream Finetuning of SMART. As discussed in Section 8.3, the pretrained representation can be used to learn policies for different tasks. To do so, we append a policy head π

[2]The implementation details are provided in Section 8.7.1.2.

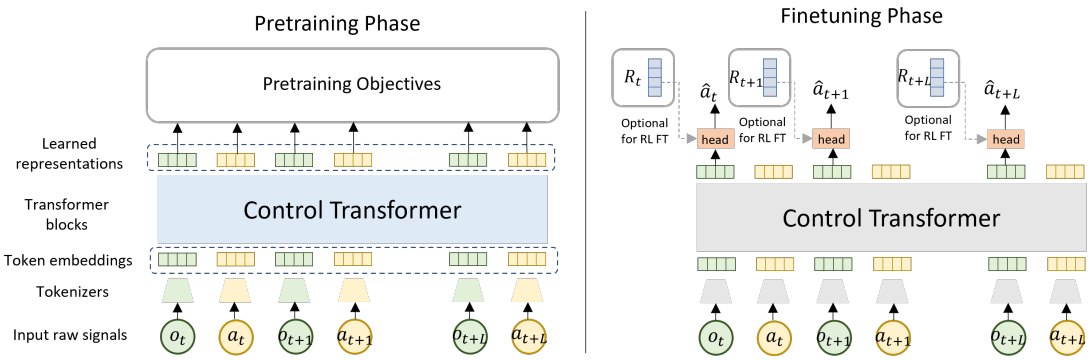


Figure 8.1: Architecture of Control Transformer. In the pretraining phase, we use the control-centric objective introduced in Section 8.4.2 to train representation over multiple tasks; in the finetuning phase where a specific task is given, we learn a policy based on the pretrained representation (pretrained weights are shown in grey blocks). The construction of the policy head can vary for different downstream datasets or learning methods.

to the observation representation, such that $\pi(\phi(o_t))$ predicts the proper action for observation o_t. We can train the policy head using both IL and RL. For our IL experiments we use behavior cloning with expert demonstrations, where we feed $\phi(o_t)$ into a policy head to get action predictions. For RL we use a return-to-go (RTG)-conditioned policy with trajectories that contain reward values. We feed $\phi(o_t)$ along with an RTG embedding to get the policy head's action predictions. Online RL with transformer-based models is still a novel field and is not the focus of this work. But our pretraining method can also be combined with online RL finetuning such as [227].

Comparison with Prior Decision-making Transformers. Recent works leverage transformer architectures for modeling sequential decision making problems [208, 209, 224] as summarized in Section 8.2. Most of these models use reward information in the input sequence, as their goal is to directly learn a policy for a specific task. In contrast, our goal is to pretrain representations for various downstream tasks, and thus our CT uses reward-free control sequence as the model input; rewards are only used for downstream task when needed. There are several

benefits of making representation agnostic to reward during pretraining. **(1)** A pretrained model requiring reward input does not flexibly fit some downstream learning scenarios such as behavior cloning, while CT can be a unified model for various learning methods. A user can easily learn a policy under different learning methods (IL or RL) without modifying transformer blocks. **(2)** Reward distribution can be significantly different when the task or policy changes, making a reward-dependent representation less resilient to distribution shift. We show in Section 8.5.3 that *utilizing reward during pretraining may hurt the overall downstream performance*.

8.4.2 Control-centric Self-supervised Pretraining Objectives

Our pretraining objective employs three terms: forward dynamics prediction, inverse dynamics prediction, and random masked hindsight control. The first two terms focus on local and short-term dynamics, while the third term is designed to capture more global and long-term temporal dependence. As motivated in Section 8.4.1, these terms are based on control sequences and are reward-free, such that they can be used for multiple tasks. Figure 8.2 illustrates each objective.

I. Forward Dynamics Prediction. For each observation-action pair (o_t, a_t) in a control sequence, we aim to predict the next immediate latent state. Let g be the observation tokenizer being trained. We maintain a momentum encoder $\bar{g}$ as the exponential moving average of g, to generate the target latent state from the next observation o_{t+1}, i.e., $\hat{s}_{t+1} := \bar{g}(o_{t+1})$. The idea of momentum encoder is widely used when the target value is not fixed [1, 203], in order to make training more stable. Then, we train a linear head to predict $\hat{s}_{t+1}$ based on $(\phi(o_t), \phi(a_t))$. The

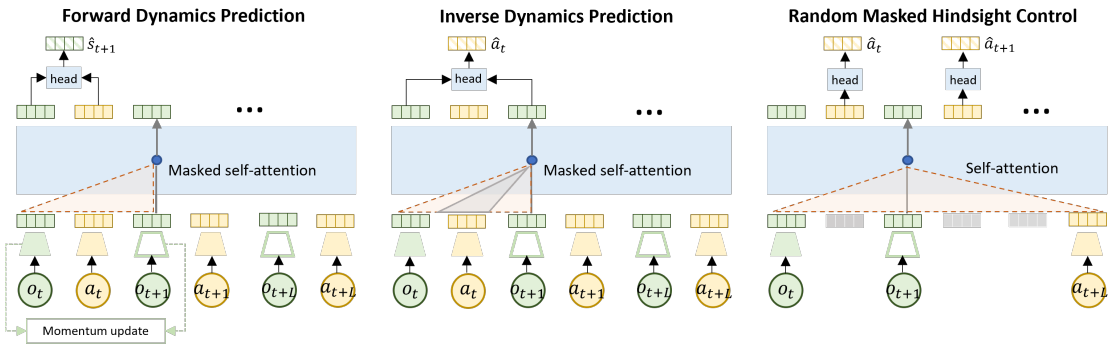


Figure 8.2: The three terms of our proposed pretraining objective. These terms use different attention masks, while the weights of transformer and encoders are shared and jointly optimized. The red shaded areas denote the attention span, while the grey regions are masked.

forward prediction loss is defined as:

$$L_{\text{fwd}} := \text{MSE}\left(f_{\text{fwd}}(\phi(o_t), \phi(a_t)), \hat{s}_{t+1}\right), \tag{8.1}$$

where f_{fwd} is a linear prediction head. This forward prediction captures the local transition information in the embedding space.

II. Inverse Dynamics Prediction. For each consecutive observation pair (o_t, o_{t+1}), we learn to recover the action that leads o_t to o_{t+1}, which gives the loss function

$$L_{\text{inv}} := \text{MSE}\left(f_{\text{inv}}(\phi(o_t), \phi(o_{t+1})), a_t\right), \tag{8.2}$$

where f_{inv} is a linear prediction head. Note that in a causal transformer, a_t is visible to the model when generating representation of o_{t+1} which can lead to trivial solutions, so we modify the original causal mask and mask out a_t from the attention of o_{t+1}. Therefore, the observation representation is forced to contain information for relevant actions and transitions.

Both forward and inverse predictions focus on local dynamics induced by the transition kernel P. However, fitting local dynamics only may result in collapsed representation, i.e., the model learns the same representation for two semantically different observations [231]. To perform well in downstream tasks, long-term temporal dependence should also be captured in the

representation. We achieve this by a novel random masked hindsight control term in pretraining.

III. Random Masked Hindsight Control. Given a control sequence $h = (o_t, a_t, \cdots, o_{t+L}, a_{t+L})$, we randomly mask k actions and k' observations, and recover the masked actions based on the remaining incomplete sequence. This idea of masked token prediction is related to BERT [201] for language modeling, but note that we only predict the masked actions for the purpose of control. More rationale behind the selection of k and k' is explained in Section 8.4.2. This objective is akin to asking the question "what actions should I take to generate such a trajectory?" Therefore, we replace the causal attention mask with a non-causal one, to temporarily allow the model "see the future", as shown in Figure 8.2(right). Let $\tilde{h}$ be the masked control sequence, and $0 \leq s_1, s_2, \cdots, s_k \leq L$ be the selected indices for masked actions, then the loss function can be defined as:

$$L_{\text{mask-inv}} := \sum\nolimits_{i=1}^{k} \text{MSE}\left(f_{\text{mask-inv}}(\phi_M(\mathbf{m}_{t+s_i}); \tilde{h}), a_{t+s_i}\right), \tag{8.3}$$

where $f_{\text{mask-inv}}$ is a linear prediction head, and ϕ_M is the transformer model without a causal mask. As a result, we encourage the model to learn controllable representations and global temporal relations, and to attend to the most essential representations for multi-step control. The idea of our random masked hindsight control is also related to the multi-step inverse prediction proposed by a concurrent work [232], which predicts a_t given s_t and s_{t+l} for a random interger l and theoretically shows the effectiveness of this method in discovering controllable states. Our random masked hindsight control is different as it predicts multiple actions altogether from randomly masked sequences with a transformer model, which can efficiently learn the control information in large-scale tasks, and avoid ambiguity caused by different paths between states.

Finally, our pretraining objective is the summation of the above three terms with equal

weights, i.e.,

$$\min_{\phi, f_{\text{fwd}}, f_{\text{inv}}, f_{\text{mask-inv}}} L_{\text{fwd}} + L_{\text{inv}} + L_{\text{mask-inv}}. \tag{8.4}$$

We use equal weights for these objectives, which in experiments renders good performance.

8.5 Empirical Results

We provide empirical results to demonstrate the effectiveness of our proposed pretraining method, while aiming to answer the following questions: **(1)** Can SMART effectively improve learning efficiency and performance in a variety of downstream tasks under different learning methods? **(2)** How well can SMART generalize to out-of-distribution tasks? **(3)** Is SMART resistant to low-quality pretraining data? **(4)** How does SMART compare to state-of-the-art pretraining techniques? **(5)** How do different pretraining objectives affect the downstream performance?

8.5.1 Experimental Setup

We evaluate SMART on the DeepMind Control (DMC) suite [199], which contains a series of continuous control tasks with RGB image observations. There are multiple domains (physical models with different state and action spaces) and multiple tasks (associated with a particular MDP) within each domain, which creates diverse scenarios for evaluating pretrained representations. Our experiments use 10 different tasks spanning over 6 domains. In pretraining, we use an offline dataset collected over 5 tasks, while the other 5 tasks (with 2 unseen domains) are held out to test the generalizability of SMART. A full list of tested domains and tasks is in Section 8.7.1.1.

Pretraining Tasks and Datasets. We pretrain SMART on 5 tasks:

- `Random`: Trajectories with random environment interactions, with 400K timesteps per task.

- `Exploratory`: Trajectories generated in the exploratory stage of multiple RL agents with different random seeds, with 400K timesteps per task.

Downstream Tasks, Learning Methods and Datasets. We evaluate the pretrained models in the 5 seen tasks and another 5 unseen tasks: cartpole-balance, hopper-stand, walker-walk, pendulum-swingup and finger-spin (the last two tasks are from unseen domains with different state-action spaces). We consider two learning methods: return-to-go-conditioned policy learning (RTG) and behavior cloning (BC). For RTG, we use the `Sampled Replay` dataset containing randomly sampled trajectories from the full replay buffer collected by learning agent. For BC, we use the `Expert` trajectories with the highest returns from the full replay buffer of learning agents. The downstream dataset for every task only has 100K timesteps, making it challenging to learn from scratch.

Implementation Details. Our implementation of CT is based on a GPT model [233] with 8 layers and 8 attention heads. We use context length L = 30 and embedding size d = 256. As explained in Section 8.4.2, k and k' are linearly increased from 1 to L and $L/2$, respectively. The observation tokenizer is a standard 3-layer CNN. Action tokenizer, return tokenizer, and all single-layer linear prediction heads. We found that freezing the pretrained weights in downstream tasks works well in relatively simple environments, but fails in harder ones. Therefore, we fine-tune the entire model including transformer blocks for all downstream tasks. Since actions in different domains have different dimensions and physical meanings, we project the raw actions into a larger common action space to train the action tokenizer. When there is a novel down-

stream task with a different action space, we simply re-initialize the action tokenizer and finetune it. Please see Section 8.7.1.2 for more implementation and hyperparameter details.

Baselines. We compare SMART with the following transformer-based pretraining baselines:

- `Scratch` trains a policy with randomly initialized CT representation weights.
- `ACL` [211] is a modified BERT [201] that randomly masks and predicts tokens with a contrastive loss, pretrained on the same dataset as SMART.
- `DT` [209] pretrained on the same dataset as ours but uses extra reward supervision.
- `CT-single` is a variant of SMART, which pretrains CT with a single-task dataset containing trajectories from the downstream environment.

For fair comparisons, we use the same network architecture for the baseline models (except for DT where we keep their original network structure with RTG as transformer inputs) and train them with the same configurations. We also compare SMART with other state-of-the-art pretraining works, such as `CPC` [210] and `ATC` [191], using the same pretraining-finetuning pipeline. However, as these approaches are built upon ResNet backbones, a direct comparison of a Transformer against a ResNet could be not straightforward. Our preliminary experiments show that transformer models have comparable or better performance than ResNet-based models.

Evaluation Metrics. To evaluate the quality of pretrained representations we report the average cumulative reward obtained in downstream tasks after finetuning. We deploy the trained policies in each environment for 50 episodes and report average returns.

For both BC and RTG downstream learning, we report the average cumulative reward of the

learned policy by interacting with the environment for 50 episodes. In Figure 8.5, we calculate the *normalized reward* based on expert scores.

In Figure 8.6 and Figure 8.7a, we report the relative improvement of each ablated method calculated by the following formula.

$$\text{relative\_improvement} := \frac{\text{method\_reward} - \text{scratch\_reward}}{\text{scratch\_reward}}, \tag{8.5}$$

where the scratch_reward is the best reward of training from scratch using the same learning configurations.

8.5.2 Experimental Results

We first evaluate the **versatility** of SMART: 1) whether a single pretrained model can be finetuned with different downstream learning methods (i.e. RTG and BC); and 2) whether the pretrained model can adapt towards various downstream tasks. Figure 8.3 compares the the reward curve of SMART with `Scratch` and `CT-Single`, where models are pretrained with `Exploratory` dataset[3]. To avoid overlapping of curves, comparison with `ACL` and `DT` is shown and discussed later in Figure 8.5. It can be seen that pretrained CT from both single-task dataset (`CT-single`) and multi-task dataset (SMART) can achieve much better results than training from scratch. In general, under both RTG and BC finetuning, pretrained models have a warm start, a faster convergence rate, and a relatively better asymptotic performance in a variety of downstream tasks. In most cases, pretraining CT from multi-task dataset (SMART) yields better results than pretraining with only in-task data (`CT-single`), although it is harder to accommodate multiple different tasks with the same model capacity, which suggests that SMART can extract common knowledge from diverse tasks.

[3]Models pretrained with `Random` dataset show similar results, as can be seen in Section 8.7.2

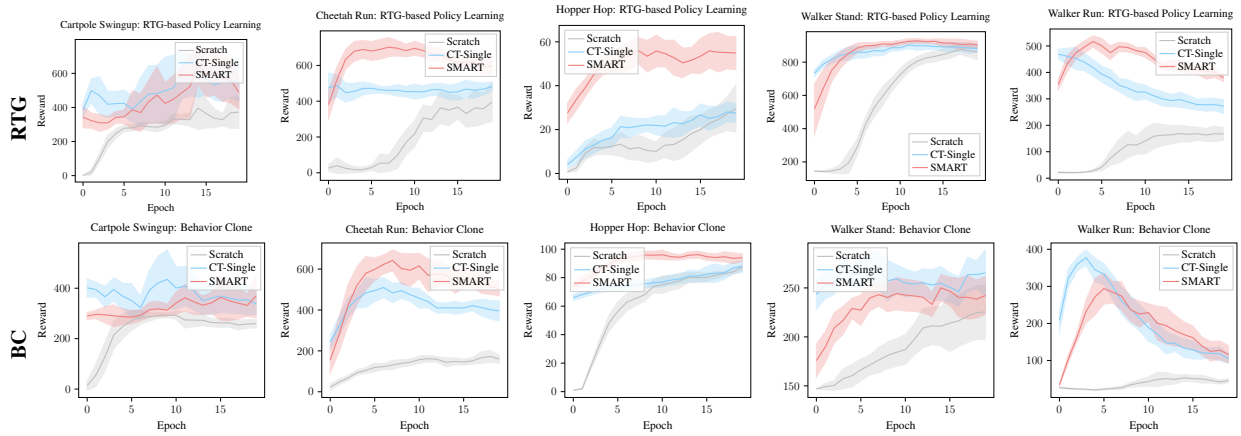


Figure 8.3: Downstream learning rewards of SMART (red) compared with pretraining CT with single-task data (blue) and training from scratch (gray). Results are averaged over 3 random seeds.

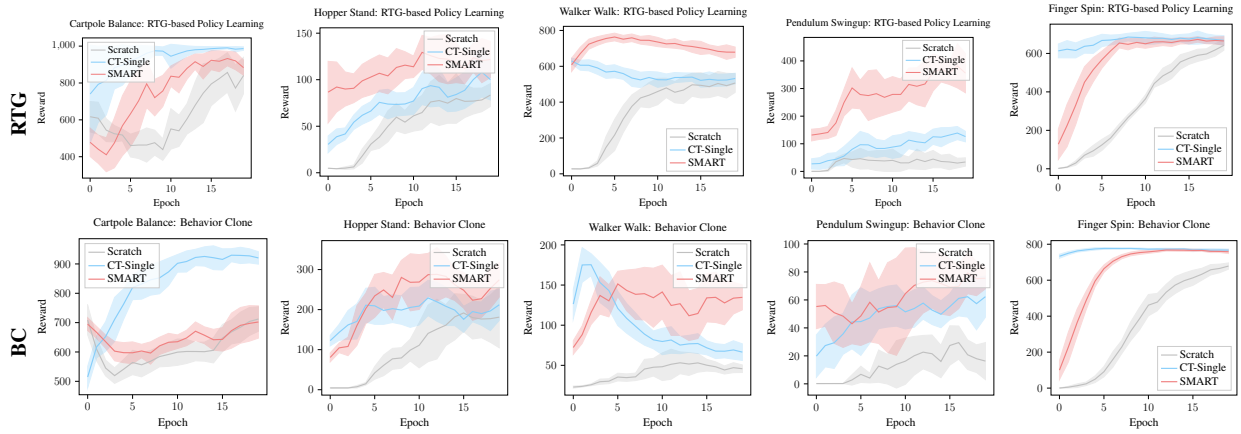


Figure 8.4: Downstream learning rewards **in unseen tasks and domains** of SMART (red) compared with pretraining CT with single-task data (blue) and training from scratch (gray). Results are averaged over 3 seeds.

Next, we show the **generalizability** of SMART. Figure 8.4 shows the performance of SMART pretrained on `Exploratory` dataset[3], compared to `Scratch` and `CT-single` on 5 unseen tasks. We can see that SMART is able to generalize to unseen tasks and even unseen domains, whose distributions have a larger discrepancy as compared to the pretraining dataset. Surprisingly, SMART achieves better performance than `CT-single` in most tasks, even though `CT-single` has already seen the downstream environments. This suggests that good gener-

alization ability can be obtained from learning underlying information which might be shared among multiple tasks and domains, spanning a diverse set of distributions.

Next we evaluate the **resilience** of SMART by comparing with all aforementioned baselines, as visualized in Figure 8.5. We aggregate the results in all tasks by averaging the normalized reward (dividing raw scores by expert scores) in both RTG and BC settings. When using the `Exploratory` dataset for pretraining, SMART outperforms `ACL`, and is comparable to `DT` which has extra information of reward. When pretrained with the `Random` dataset, SMART is significantly better than `DT` and `ACL`, while `ACL` fails to outperform training from scratch. This result show that SMART is robust to low-quality data as compared to other baseline methods.

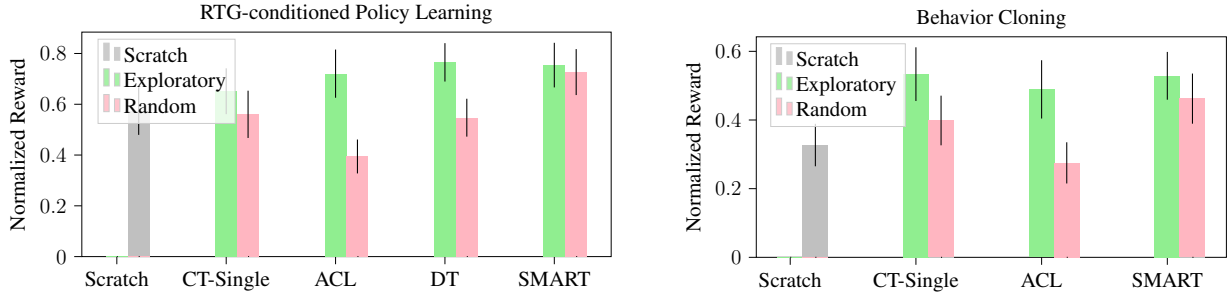


Figure 8.5: Downstream learning rewards (normalized by expert score) of all methods using `Exploratory` and `Random` dataset. The gap between each pair of green and red bars corresponds to the resilience of each method to pretraining data quality, and our SMART shows the best resilience among all baselines.

8.5.3 Ablation and Discussion

Ablation of Pretraining Objectives. As we analyzed in Section 8.4.2, forward prediction and inverse prediction aim to learn the information of short-term control, while the random masked hindsight control (in short, Mask-Ctl) learns the long-term control information. To show the effectiveness of combining these two kinds of information, we conduct ablation study over these three terms. Figure 8.6 demonstrates their relative improvements wrt `Scratch`. We can

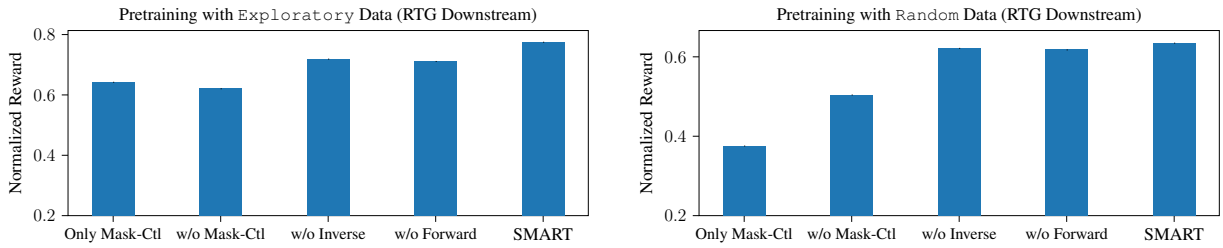
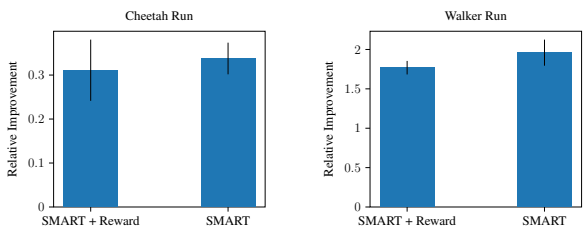
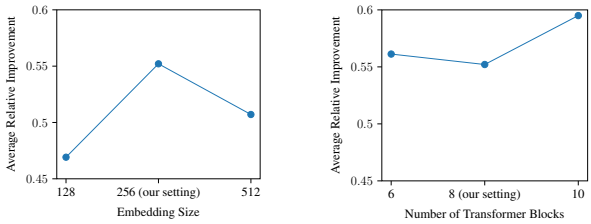


Figure 8.6: Ablation study on proposed pretraining objective. Rewards are averaged over tasks. Both long-term control information (Mask-Ctl) and short-term control information (Forward and Inverse) are important.



(a) Reward-based v.s. reward-free pretraining.



(b) Overall performance v.s. model capacity.

Figure 8.7: Ablation study of the effect of reward in pretraining and comparison of various model capacity.

see that only learning long-term control information (Only Mask-Ctl) or only learning short-term control information (w/o Mask-Ctl) renders much lower performance than the original SMART that leverages both types of information. In addition, we find that both the performance slightly drops if either forward prediction or inverse prediction is removed, as the combination of these two terms leads to more stable and comprehensive representation for short-term control. This ablation study verifies the effectiveness of each component in the proposed pretraining objective.

Reward-free v.s. Reward-based Pretraining. As discussed in Section 8.4.1, including reward information in pretraining objectives is not necessarily helpful. We study the effects of rewards in pretraining by adding two reward-based objectives in the pretraining phase: immediate reward prediction and RTG-based action prediction. We evaluate this reward-based variant of SMART using exploratory dataset and RTG-conditioned downstream learning. Note that the RTG-based action prediction is used both in pretraining and finetuning of the reward-based vari-

ant, providing strong supervision for the pretrained model. However, we (surprisingly) observe that such supervision does not improve the downstream performance in many tasks, as shown in Figure 8.7a. A potential reason is that reward-based objectives are more fragile to distribution shifts, which also explains the non-ideal performance of DT pretrained from random data.

Discussion on Model Capacity. In large-scale training problems, performance usually benefits from larger model capacity [234]. We investigate if this also applies to sequential decision making tasks by varying the embedding size (width) and the number of layers (depth) in CT. The aggregated results averaged over all tasks are show in Figure 8.7b. From the comparison, we can see that in general, increasing the model depth leads to a better performance. However, when embedding size gets too large, the performance further drops, as a large representation space might allow for irrelevant information.

8.6 Conclusion

This chapter studies how to pretrain a versatile, generalizable and resilient representation model for multi-task sequential decision making. We propose a self-supervised and control-centric objective that encourages the transformer-based model to capture control-relevant representation. Empirical results in multiple domains and tasks demonstrate the effectiveness of the proposed method, as well as its robustness to distribution shift and low-quality data. Future work includes strengthen the attention mechanism on both spatial observation space and temporal state-observation interactions, as well as investigating its potential generalization in a wider range of application scenarios.

8.7 Supplemental Materials: Additional Details and Experiments

8.7.1 Experiment Setting Details

8.7.1.1 Environment and Dataset.

Table 8.2 lists all domains and tasks from DMC used in our experiments, and their relations are further depicted in Figure 8.8.

Phase	Domain	Task	Expert Score by SAC
Pretraining & Finetuning	cartpole	swingup	875
	hopper	hop	200
	cheetah	run	850
	walker	stand	980
	walker	run	700
Finetuning only	cartpole	balance	1000
	hopper	stand	900
	walker	walk	950
	pendulum	swingup	1000
	finger	spin	800

Table 8.2: A list of domains and tasks used in pretraining and finetuning. The first 5 tasks are used for pretraining. In the finetuning phase, we use the pretrained model to learn policies in all 10 tasks, including the last 5 tasks that are unseen during pretraining.

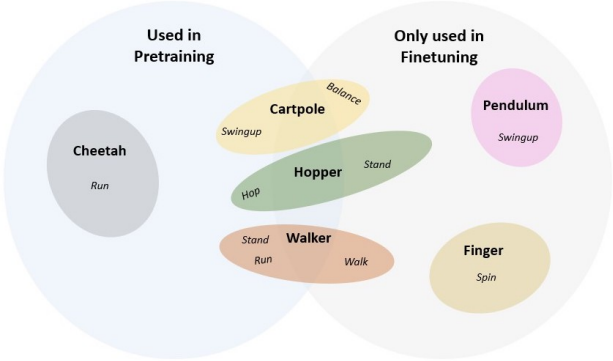


Figure 8.8: Graphical relations of all tasks involved.

To generate our datasets for pretraining and finetuning, we first train 5 agents (correspond-

ing to different random seeds) for each task with ground truth vector states using SAC [98] for 1M steps, and collect the full replay buffer with corresponding RGB images ($3\times84\times84$) rendered by the physical simulator. Then, we divide the replay buffers and create the following datasets of different qualities.

- `Random`: Randomly generated interaction trajectories. This dataset has 400K timesteps per task.
- `Exploratory`: The first 80K timesteps of each SAC learner, corresponding to the exploratory stage. For all 5 agents, this leads to a cumulative of 400K timesteps per task.

Note that different tasks have different difficulties for the SAC agent to converge. For example, in cartpole, the agent converges with much less samples than in walker. Therefore, we slightly adjust the proportion of data from different tasks in multi-task pretraining to avoid overfitting to simple tasks and underfitting to harder tasks. In pretraining, we use 40K timesteps from each cartpole behavior agent (200K by 5 agents), and 90K timesteps from each walker behavior agent (450K by 5 agents), and 80K for all other agents (400K by 5 agents).

Datasets for downstream learning:

- `Sampled Replay` (for RTG): We randomly sample 10% trajectories from the full replay buffer of 1 SAC agent, resulting in a dataset of size 100K per task, with diverse return distribution.
- `Expert` (for BC): We select 10% trajectories with the highest returns from the full replay buffer of 1 SAC agent, resulting in an expert dataset of size 100K per task.

8.7.1.2 Model and Hyperparameters.

Following the implementation of Decision Transformer [209], our transformer backbone is based on the minGPT implementation https://github.com/karpathy/minGPT with the default AdamW optimizer [235]. For both pretraining and finetuning, the learning rate is set to be 6×10^{-4} and batch size 256. For learning rate, linear warmup and cosine decay are used. A context length 30 is used in all tasks for both training and execution. We tested different context lengths in preliminary experiments, and a shorter context length (5/10/20) does not work as well as 30 when training from scratch. We use 8 attention heads, 8 layer attention blocks, and an embedding size 256 in all experiments, for both our model and baselines. There are 10.8 M trainable parameters in the model. We test SMART with varying layer numbers and embedding sizes in Section 8.5.3.

For the execution of learned RTG-conditioned policies, we set the expected RTG as the expert scores in Table 8.2. Tuning the RTG setting may further increase the results. But since our focus is to show the effectiveness of pretraining, we did not explore other possibilities.

All models are trained for 10 epochs in pretraining, and 20 epochs for each downstream task. The performance of the best checkpoint is reported, as detailed in the next section.

8.7.2 Additional Experiment Results

Curves with the `Random` Dataset. The results in Figure 8.3 and Figure 8.4 are generated with the `Exploratory` pretraining dataset. Now, we show the performance of models pretrained using the `Random` dataset in seen tasks and unseen tasks in Figure 8.9 and Figure 8.10, respectively.

Although the single-task pretrained model consistently outperforms training from scratch

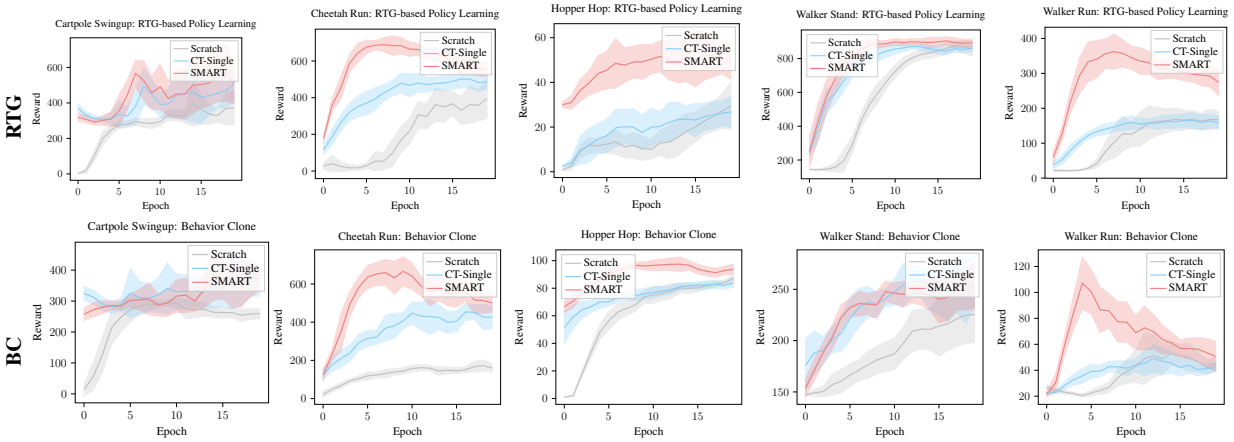


Figure 8.9: Downstream learning rewards of SMART (red) compared with pretraining CT with single-task data (blue) and training from scratch (gray), using the `Random` pretraining dataset. Results are averaged over 3 random seeds.

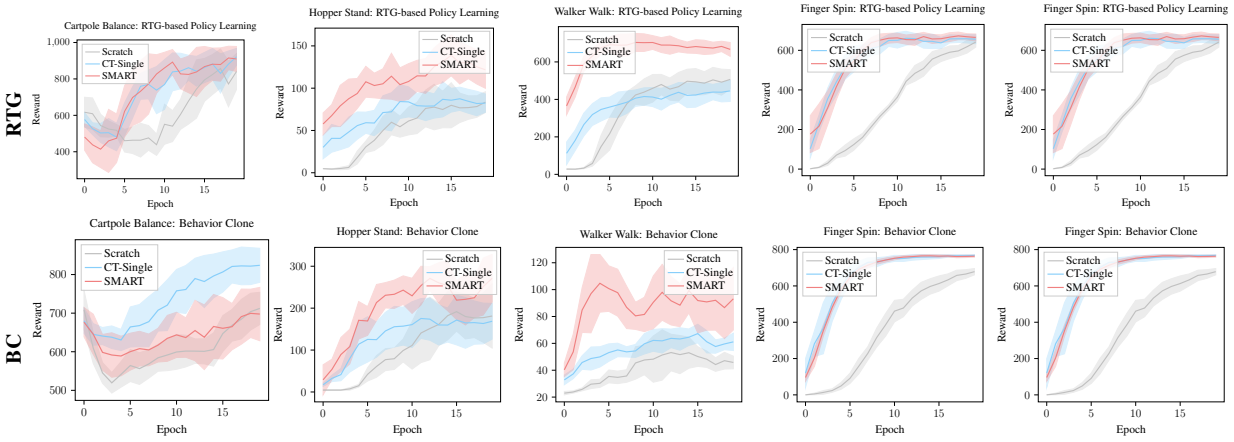


Figure 8.10: Downstream learning rewards in **unseen tasks** and domains of SMART (red) compared with pretraining CT with single-task data (blue) and training from scratch (gray), using the `Random` pretraining dataset. Results are averaged over 3 random seeds.

when pretrained with the `Exploratory` dataset, it sometime gets worst-than-scratch downstream performance when pretrained with the Random dataset. Therefore, it is challenging to overcome the distribution shift problem. In contrast, SMART still achieves much better performance than training from scratch in all tested tasks, which verifies the resilience of SMART due to multi-task self-supervised pretraining.

Chapter 9: Conclusions and Future Perspectives

9.1 Dissertation Summary

In this dissertation, we systematically investigated the robustness and adaptability of RL methods in various scenarios, proposing a series of solutions to enhance the robustness of RL and to improve the adaptability of RL.

For robustness, our major contributions are as follows:

- We introduce approaches to measure the vulnerabilities of existing RL techniques under adversarial perturbations to policy inputs during test-time (Chapter 2), adversarial perturbations to training trajectories during training-time (Chapter 4), and adversarial perturbations to interactions between different agents (Chapter 5).

- We establish a novel geometric understanding of adversarial attacks and defenses (Chapter 2, Chapter 4), which serve as the foundation of our proposed methods. We also characterize the optimality of existing attack and defense algorithms with theoretical analysis and geometric insights (Chapter 2, Chapter 4). The proposed geometric-motivated techniques can inspire and benefit future studies in the adversarial robustness of RL agents.

- We propose effective and efficient robust training paradigms to improve the robustness of RL agents with state-of-the-art empirical performance (Chapter 2, Chapter 3) and theoret-

ical guarantees (Chapter 5). With a focus on the worst-case performance, we let the agent achieve generic robustness across various types of attacks, including strong attacks and weaker ones.

For adaptability, our major contributions are as follows:

- We design algorithms to automatically discover similarities in tasks and utilize them to facilitate learning (Chapter 6, Chapter 7). These algorithms can let the agent learn by analogy without negative transfer, based on theoretical analysis and guarantees (Chapter 6, Chapter 7).

- We propose several solutions to commonly desired real-world adaptation challenges, where the state space, action space and dynamics of the environment can all change. In Chapter 6, we show how modular-level similarities can be learned and used to enable knowledge transfer across different domains. In Chapter 7 where the observation shift changes the format policy input, we utilize the similarity of latent dynamics to accelerate learning in the new task. In Chapter 8, we achieve fast adaptation to multiple seen/unseen agents and tasks, by utilizing the similarity of the mapping from visual observations to latent representations.

- We theoretically investigate the role of representation learning in reinforcement learning, demonstrating that representation can be a bridge to connect various tasks to achieve adaptability (Chapter 7, Chapter 8).

- The techniques introduced by Chapter 8 can help build foundation models for RL with transformer backbones, which takes an important step towards general artificial intelligence.

All of the above solutions proposed in this dissertation can serve as a plug-in component and be combined with existing RL algorithms to make them robust and adaptable. Therefore, the techniques and analysis introduced in this dissertation can be widely applied to many real-world sequential decision-making problems.

9.2 Future Perspectives

Towards Better Robustness and Adaptability: Pushing the Boundaries. It is important for future research on the robustness and adaptability of RL agents to provide more general and theoretically grounded solutions to real-world problems. Some open problems include: (1) characterizing the trade-off between natural performance and robustness of RL agents, (2) developing RL approaches with certifiable robustness in both the training phase and the deployment phase, (3) investigating the robustness of RL models under a wider range of threat models, (4) pretraining generic foundation models from multiple modalities that can be applied to a broad range of decision-making problems, and (5) designing lifelong learning agents to fulfill the needs of multitasking.

Towards Real-life Applications: Research for Social Good. As motivated in Chapter 1, this dissertation is motivated by the need of more robust and more adaptable sequential decision-making models in the real world. From an applied perspective, another important future direction is to improve real-life decision-making systems, such as autonomous driving, home robots, healthcare systems, etc. These applications can benefit society and bring convenience to people with disabilities and diseases. RL approaches, in combination with other artificial intelligence techniques, have huge potential to build more intelligent systems by interacting with people and

the environment. Robustness and adaptability are crucial in these high-stakes applications, to make sure that the trained agents can not only work for seen scenarios, but also guarantee safe behaviors under unexpected inputs and quickly produce good behaviors when changes happen.

Bibliography

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

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[3] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[4] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[5] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[6] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[7] Yanchao Sun, Da Huo, and Furong Huang. Vulnerability-aware poisoning mechanism for online rl with unknown dynamics. In *International Conference on Learning Representations*, 2021.

[8] Yanchao Sun, Ruijie Zheng, Yongyuan Liang, and Furong Huang. Who is the strongest enemy? towards optimal and efficient evasion attacks in deep RL. In *International Conference on Learning Representations*, 2022.

[9] Yongyuan Liang, Yanchao Sun, Ruijie Zheng, and Furong Huang. Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[10] Yanchao Sun, Ruijie Zheng, Parisa Hassanzadeh, Yongyuan Liang, Soheil Feizi, Sumitra Ganesh, and Furong Huang. Certifiably robust policy learning against adversarial multi-agent communication. In *The Eleventh International Conference on Learning Representations*, 2023.

[11] Yanchao Sun and Furong Huang. Can agents learn by analogy? an inferable model for pac reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1332–1340, 2020.

[12] Yanchao Sun, Xiangyu Yin, and Furong Huang. Temple: Learning template of transitions for sample efficient multi-task rl. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9765–9773, 2021.

[13] Yanchao Sun, Ruijie Zheng, Xiyao Wang, Andrew E Cohen, and Furong Huang. Transfer RL across observation feature spaces via model-based regularization. In *International Conference on Learning Representations*, 2022.

[14] Yanchao Sun, Shuang Ma, Ratnesh Madaan, Rogerio Bonatti, Furong Huang, and Ashish Kapoor. SMART: Self-supervised multi-task pretraining with control transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013.

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

[17] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 2040–2042, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[18] Huan Zhang, Hongge Chen, Duane S Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. In *International Conference on Learning Representations*, 2021.

[19] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21024–21037. Curran Associates, Inc., 2020.

[20] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[21] Ezgi Korkmaz. Nesterov momentum adversarial perturbations in the deep reinforcement learning domain. In *ICML 2020 Inductive Biases, Invariances and Generalization in Reinforcement Learning Workshop*, 2020.

[22] Alessio Russo and Alexandre Proutiere. Optimal attacks on reinforcement learning policies. In *American Control Conference (ACC).*, 2021.

[23] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 3756–3762. AAAI Press, 2017.

[24] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[25] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

[26] Matthew Inkawhich, Yiran Chen, and Hai Li. Snooping attacks on deep reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 557–565, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[27] Chaowei Xiao, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Mingyan Liu, Bo Li, and Dawn Song. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019.

[28] Kai Liang Tan, Yasaman Esfandiari, Xian Yeow Lee, Soumik Sarkar, et al. Robustifying reinforcement learning agents via action space adversarial training. In *2020 American control conference (ACC)*, pages 3959–3964. IEEE, 2020.

[29] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pages 6215–6224. PMLR, 2019.

[30] Xian Yeow Lee, Yasaman Esfandiari, Kai Liang Tan, and Soumik Sarkar. Query-based targeted action-space adversarial policies on deep reinforcement learning agents. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, ICCPS '21, page 87–97, New York, NY, USA, 2021. Association for Computing Machinery.

[31] Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 262–275. Springer, 2017.

[32] Yunhan Huang and Quanyan Zhu. Deceptive reinforcement learning under adversarial manipulations on cost signals. In *International Conference on Decision and Game Theory for Security*, pages 217–237. Springer, 2019.

[33] Yanchao Sun, Da Huo, and Furong Huang. Vulnerability-aware poisoning mechanism for online rl with unknown dynamics. In *International Conference on Learning Representations*, 2021.

[34] Xuezhou Zhang, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. Adaptive reward-poisoning attacks against reinforcement learning. In *International Conference on Machine Learning*, 2020.

[35] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on Machine Learning*, pages 7974–7984, 2020.

[36] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

[37] Marc Fischer, Matthew Mirman, Steven Stalder, and Martin Vechev. Online robustness training for deep reinforcement learning. *arXiv preprint arXiv:1911.00887*, 2019.

[38] Björn Lütjens, Michael Everett, and Jonathan P How. Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning*, pages 1328–1337. PMLR, 2020.

[39] Tuomas Oikarinen, Tsui-Wei Weng, and Luca Daniel. Robust deep reinforcement learning through adversarial loss. *arXiv preprint arXiv:2008.01976*, 2020.

[40] Robert Dadashi, Adrien Ali Taiga, Nicolas Le Roux, Dale Schuurmans, and Marc G. Bellemare. The value function polytope in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1486–1495, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[41] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[42] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

[43] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.

[44] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[45] David Silver Hado Van Hasselt, Arthur Guez. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[46] Ioannis Antonoglou Tom Schaul, John Quan and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

[47] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

[48] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[49] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *CoRR*, abs/1810.00069, 2018.

[50] Qianli Shen, Yan Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. Deep reinforcement learning with robust and smooth policy. In *International Conference on Machine Learning*, pages 8707–8718. PMLR, 2020.

[51] Ezgi Korkmaz. Investigating vulnerabilities of deep neural policies. In *Uncertainty in Artificial Intelligence*, pages 1661–1670. PMLR, 2021.

[52] Vahid Behzadan and Arslan Munir. Whatever does not kill deep reinforcement learning, makes it stronger. *CoRR*, abs/1712.09344, 2017.

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

[54] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *AAMAS*, pages 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

[55] Aounon Kumar, Alexander Levine, and Soheil Feizi. Policy smoothing for provably robust reinforcement learning. In *International Conference on Learning Representations*, 2022.

[56] Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li. CROP: certifying robust policies for reinforcement learning through functional smoothing. In *International Conference on Learning Representations*, 2022.

[57] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[58] Matthias Heger. Consideration of risk in reinforcement learning. In *International Conference on Machine Learning*, 1994.

[59] Chris Gaskett. Reinforcement learning under circumstances beyond its control. In *International Conference on Computational Intelligence for Modelling Control and Automation*, 2003.

[60] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[61] Sarah Bechtle, Yixin Lin, Akshara Rai, Ludovic Righetti, and Franziska Meier. Curious ilqr: Resolving uncertainty in model-based rl. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 162–171. PMLR, 30 Oct–01 Nov 2020.

[62] Garrett Thomas, Yuping Luo, and Tengyu Ma. Safe reinforcement learning by imagining the near future. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[63] Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. *Advances in Neural Information Processing Systems*, 26:701–709, 2013.

[64] Daniel J. Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Yuanyuan Shi, Jackie Kay, Todd Hester, Timothy Mann, and Martin Riedmiller. Robust reinforcement learning for continuous control with model misspecification. In *International Conference on Learning Representations*, 2020.

[65] Robert Dadashi, Adrien Ali Taiga, Nicolas Le Roux, Dale Schuurmans, and Marc G Bellemare. The value function polytope in reinforcement learning. In *International Conference on Machine Learning*, pages 1486–1495. PMLR, 2019.

[66] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[67] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3578–3586. PMLR, 10–15 Jul 2018.

[68] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[69] Sven Gowal, Krishnamurthy Dj Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4842–4851, 2019.

[70] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4944–4953, Red Hook, NY, USA, 2018. Curran Associates Inc.

[71] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[72] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.

[73] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33, 2020.

[74] Aviv Tamar, Huan Xu, and Shie Mannor. Scaling up robust mdps by reinforcement learning. *CoRR*, abs/1306.6189, 2013.

[75] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *CoRR*, abs/1712.09665, 2017.

[76] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[77] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

[78] Saul B Gelfand and Sanjoy K Mitter. Recursive stochastic algorithms for global optimization in r^d. *SIAM Journal on Control and Optimization*, 29(5):999–1018, 1991.

[79] Arthur Guez and H Van Hasselt. Deep reinforcement learning with double q-learning. *Association for the Advancement of Artificial Intelligence*, 2015.

[80] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[81] David Silver Hado Van Hasselt, Arthur Guez. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[82] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

[83] Tong Chen, Jiqiang Liu, Yingxiao Xiang, Wenjia Niu, Endong Tong, and Zhen Han. Adversarial attack and defense in reinforcement learning-from ai security view. *Cybersecurity*, 2(1):11, 2019.

[84] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

[85] Yizhen Wang and Kamalika Chaudhuri. Data poisoning attacks against online learning. *arXiv preprint arXiv:1808.08994*, 2018.

[86] Yuzhe Ma, Xuezhou Zhang, Wen Sun, and Jerry Zhu. Policy poisoning in batch reinforcement learning and control. In *Advances in Neural Information Processing Systems*, pages 14543–14553, 2019.

[87] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. Market making via reinforcement learning. *CoRR*, abs/1804.04216, 2018.

[88] Haoqi Zhang and David C Parkes. Value-based policy teaching with active indirect elicitation. In *AAAI*, volume 8, pages 208–214, 2008.

[89] Haoqi Zhang, David C Parkes, and Yiling Chen. Policy teaching through reward function learning. In *Proceedings of the 10th ACM conference on Electronic commerce*, pages 295–304, 2009.

[90] Qu Xinghua, Zhu Sun, Yew Ong, Abhishek Gupta, and Pengfei Wei. Minimalistic attacks: How little it takes to fool deep reinforcement learning policies. *IEEE Transactions on Cognitive and Developmental Systems*, PP:1–1, 02 2020.

[91] T. T. Nguyen, C. Soussen, J. Idier, and E. Djermoune. Np-hardness of l0 minimization problems: revision and extension to the non-negative setting. In *2019 13th International conference on Sampling Theory and Applications (SampTA)*, pages 1–4, 2019.

[92] Julien Perolat, Bruno Scherrer, Bilal Piot, and Olivier Pietquin. Approximate dynamic programming for two-player zero-sum markov games. volume 37 of *Proceedings of Machine Learning Research*, pages 1321–1329, Lille, France, 07–09 Jul 2015. PMLR.

[93] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[94] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[95] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[96] Yizhen Wang, Somesh Jha, and Kamalika Chaudhuri. Analyzing the robustness of nearest neighbors to adversarial examples. *arXiv preprint arXiv:1706.03922*, 2017.

[97] Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.

[98] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[99] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[100] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[101] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[102] Matthew John Hausknecht. *Cooperation and communication in multiagent deep reinforcement learning*. PhD thesis, 2016.

[103] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.

[104] Swapnil Naik and Vikas Maral. Cyber security — iot. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 764–767, 2017.

[105] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. *CoRR*, abs/1901.08573, 2019.

[106] Jan Blumenkamp and Amanda Prorok. The emergence of adversarial communication in multi-agent reinforcement learning, 2020.

[107] Wanqi Xue, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, and Chai Kiat Yeo. Mis-spoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, page 1418–1426, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems.

[108] Rupert Mitchell, Jan Blumenkamp, and Amanda Prorok. Gaussian process based message filtering for robust multi-agent cooperation in the presence of adversarial communication, 2020.

[109] Wenxiao Wang, Alexander Levine, and Soheil Feizi. Lethal dose conjecture on data poisoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[110] Alexander Levine and Soheil Feizi. Robustness certificates for sparse adversarial attacks by randomized ablation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4585–4593, Apr. 2020.

[111] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[112] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *Advances in Neural Information Processing Systems*, 31, 2018.

[113] Jamie Hayes. Extensions and limitations of randomized smoothing for robustness guarantees. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 786–787, 2020.

[114] Thomy Phan, Lenz Belzner, Thomas Gabor, Andreas Sedlmeier, Fabian Ritz, and Claudia Linnhoff-Popien. Resilient multi-agent reinforcement learning with adversarial value decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11308–11316, 2021.

[115] Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li. Crop: Certifying robust policies for reinforcement learning through functional smoothing. In *International Conference on Learning Representations*, 2022.

[116] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems*, 31, 2018.

[117] Murtaza Rangwala and Ryan Williams. Learning multi-agent communication through structured attentive reasoning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10088–10098. Curran Associates, Inc., 2020.

[118] Lei Yuan, Jianhao Wang, Fuxiang Zhang, Chenghe Wang, ZongZhang Zhang, Yang Yu, and Chongjie Zhang. Multi-agent incentive communication via decentralized teammate modeling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9466–9474, Jun. 2022.

[119] Woojun Kim, Jongeui Park, and Youngchul Sung. Communication in multi-agent reinforcement learning: Intention sharing. In *International Conference on Learning Representations*, 2021.

[120] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR, 2019.

[121] Yen-Cheng Liu, Junjiao Tian, Chih-Yao Ma, Nathan Glaser, Chia-Wen Kuo, and Zsolt Kira. Who2com: Collaborative perception via learnable handshake communication. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6876–6883. IEEE, 2020.

[122] James Tu, Tsunhsuan Wang, Jingkang Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. Adversarial attacks on multi-agent communication. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7768–7777, October 2021.

[123] Frans A Oliehoek. Decentralized pomdps. In *Reinforcement Learning*, pages 471–503. Springer, 2012.

[124] Frans A Oliehoek and Christopher Amato. A concise introduction to decentralized pomdps, 2015.

[125] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

[126] Anna Harutyunyan, Tim Brys, Peter Vrancx, and Ann Nowé. Off-policy shaping ensembles in reinforcement learning. *CoRR*, abs/1405.5358, 2014.

[127] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[128] Hossein K Mousavi, Mohammadreza Nazari, Martin Takáč, and Nader Motee. Multi-agent image classification via reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5020–5027. IEEE, 2019.

[129] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

[130] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[131] Justin K. Terry, Nathaniel Grammel, Sanghyun Son, and Benjamin Black. Parameter sharing for heterogeneous agents in multi-agent reinforcement learning. *CoRR*, abs/2005.13625, 2020.

[132] Samuel Berrien. Marl classification. `https://github.com/Ipsedo/MARLClassification`, 2019.

[133] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[134] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 834–843. PMLR, 06–11 Aug 2017.

[135] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022. ACM, 2007.

[136] Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. *arXiv preprint arXiv:1309.6821*, 2013.

[137] Aditya Modi, Nan Jiang, Satinder Singh, and Ambuj Tewari. Markov decision processes with continuous side information. In *Algorithmic Learning Theory*, pages 597–618, 2018.

[138] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[139] Yao Liu, Zhaohan Guo, and Emma Brunskill. Pac continuous state online multitask reinforcement learning with identification. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 438–446, 2016.

[140] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*, pages 316–324, 2014.

[141] Fei Feng, Wotao Yin, and Lin F. Yang. How Does an Approximate Model Help in Reinforcement Learning? *arXiv e-prints*, page arXiv:1912.02986, December 2019.

[142] Andrea Tirinzoni, Riccardo Poiani, and Marcello Restelli. Sequential transfer in reinforcement learning with a generative model. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9481–9492. PMLR, 13–18 Jul 2020.

[143] David Abel, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael Littman. Policy and value transfer in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 20–29, 2018.

[144] Bethany R Leffler, Michael L Littman, and Timothy Edmunds. Efficient reinforcement learning with relocatable action models. In *AAAI*, volume 7, pages 572–577, 2007.

[145] Balaraman Ravindran and Andrew G. Barto. Smdp homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1011–1016, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[146] Eyal Even-Dar and Yishay Mansour. Approximate equivalence of markov decision processes. In *Learning Theory and Kernel Machines*, pages 581–594. Springer, 2003.

[147] Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, volume 6, pages 494–499, 2006.

[148] Emma Brunskill, Bethany R. Leffler, Lihong Li, Michael L. Littman, and Nicholas Roy. Corl: A continuous-state offset-dynamics reinforcement learner. UAI'08, page 53–61, Arlington, Virginia, USA, 2008. AUAI Press.

[149] Balaraman Ravindran and Andrew G Barto. *An algebraic approach to abstraction in reinforcement learning*. PhD thesis, University of Massachusetts at Amherst, 2004.

[150] Bethany R Leffler, Michael L Littman, Alexander L Strehl, and Thomas J Walsh. Efficient exploration with latent structure. In *Robotics: Science and Systems*, pages 81–88, 2005.

[151] Jonathan Sorg and Satinder Singh. Transfer via soft homomorphisms. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 741–748, 2009.

[152] Mahsa Asadi, Mohammad Sadegh Talebi, Hippolyte Bourel, and Odalric-Ambrym Maillard. Model-based reinforcement learning exploiting state-action equivalence. *arXiv preprint arXiv:1910.04077*, 2019.

[153] Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

[154] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, March 2003.

[155] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

[156] Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 856–863, New York, NY, USA, 2005. ACM.

[157] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[158] Alexander L Strehl, Lihong Li, and Michael L Littman. Incremental model-based learners with formal learning-time guarantees. *arXiv preprint arXiv:1206.6870*, 2012.

[159] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 10–19, 2018.

[160] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[161] Thomas P Hayes. A large-deviation inequality for vector-valued martingales. *Combinatorics, Probability and Computing*, 2005.

[162] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[163] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Incremental model-based learners with formal learning-time guarantees. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'06, page 485–493, Arlington, Virginia, USA, 2006. AUAI Press.

[164] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[165] A Strehl and M Littman. Exploration via model based interval estimation. In *International Conference on Machine Learning*. Citeseer, 2004.

[166] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[167] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[168] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

[169] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[170] Steven Bohez, Tim Verbelen, Elias De Coninck, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. Sensor fusion for robot control through deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2365–2370, 2017.

[171] Sumitra Ganesh, Nelson Vadori, Mengda Xu, Hua Zheng, Prashant Reddy, and Manuela Veloso. Reinforcement learning for market making in a multi-agent dealer market. *arXiv preprint arXiv:1911.05892*, 2019.

[172] Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(9), 2007.

[173] Timothy A Mann and Yoonsuck Choe. Directed exploration in reinforcement learning with transferred knowledge. In *European Workshop on Reinforcement Learning*, pages 59–76. PMLR, 2013.

[174] Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. Policy transfer using reward shaping. In *AAMAS*, pages 181–188, 2015.

[175] Jonathan Raiman, Susan Zhang, and Christy Dennison. Neural network surgery with sets. *arXiv preprint arXiv:1912.06719*, 2019.

[176] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[177] Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block mdps. In *International Conference on Machine Learning*, pages 11214–11224. PMLR, 2020.

[178] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

[179] Benjamin Eysenbach, Swapnil Asawa, Shreyas Chaudhari, Sergey Levine, and Ruslan Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. *arXiv preprint arXiv:2006.13916*, 2020.

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

[181] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[182] Marc Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A geometric perspective on optimal representations for reinforcement learning. *Advances in neural information processing systems*, 32:4358–4369, 2019.

[183] Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G. Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. *CoRR*, abs/2006.02243, 2020.

[184] Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pages 1–9. PMLR, 2021.

[185] Zhaohan Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *International Conference on Machine Learning*, pages 3875–3886. PMLR, 2020.

[186] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *CoRR*, abs/1907.00953, 2019.

[187] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR, 2019.

[188] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018.

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

[190] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. *arXiv preprint arXiv:2102.11271*, 2021.

[191] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pages 9870–9879. PMLR, 2021.

[192] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[193] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[194] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, Belmont, MA, 1996.

[195] Rémi Munos. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1006. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[196] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

[197] Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. *arXiv preprint arXiv:2011.03506*, 2020.

[198] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10069–10076, 2020.

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

[200] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[201] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[202] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[203] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

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

[205] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control. *arXiv preprint arXiv:2203.03580*, 2022.

[206] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. Real world robot learning with masked visual pre-training. In *6th Annual Conference on Robot Learning*.

[207] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *5th Annual Conference on Robot Learning*, 2021.

[208] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

[209] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15084–15097. Curran Associates, Inc., 2021.

[210] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[211] Mengjiao Yang and Ofir Nachum. Representation matters: offline pretraining for sequential decision making. In *International Conference on Machine Learning*, pages 11784–11794. PMLR, 2021.

[212] Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, R Devon Hjelm, Philip Bachman, and Aaron C Courville. Pretraining representations for data-efficient reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12686–12699. Curran Associates, Inc., 2021.

[213] Rutav Shah and Vikash Kumar. Rrl: Resnet as representation for reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2021.

[214] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[215] Younggyo Seo, Kimin Lee, Stephen James, and Pieter Abbeel. Reinforcement learning with action-free pre-training from videos. *arXiv preprint arXiv:2203.13880*, 2022.

[216] Saurav Kadavath, Samuel Paradis, and Brian Yao. Pretraining & reinforcement learning: Sharpening the axe before cutting the tree. *arXiv preprint arXiv:2110.02497*, 2021.

[217] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[218] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[219] Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34:18459–18473, 2021.

[220] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[221] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34, 2021.

[222] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

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

[224] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34, 2021.

[225] Rogerio Bonatti, Sai Vemprala, Shuang Ma, Felipe Frujeri, Shuhang Chen, and Ashish Kapoor. Pact: Perception-action causal transformer for autoregressive robotics pre-training. *arXiv preprint arXiv:2209.11133*, 2022.

[226] Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. In *International Conference on Learning Representations*, 2022.

[227] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 27042–27059. PMLR, 17–23 Jul 2022.

[228] Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. Transdreamer: Reinforcement learning with transformer world models. *arXiv preprint arXiv:2202.09481*, 2022.

[229] Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

[230] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[231] Kate Rakelly, Abhishek Gupta, Carlos Florensa, and Sergey Levine. Which mutual-information representation learning objectives are sufficient for control? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 26345–26357. Curran Associates, Inc., 2021.

[232] Alex Lamb, Riashat Islam, Yonathan Efroni, Aniket Didolkar, Dipendra Misra, Dylan Foster, Lekan Molu, Rajan Chari, Akshay Krishnamurthy, and John Langford. Guaranteed discovery of controllable latent states with multi-step inverse models. *arXiv preprint arXiv:2207.08229*, 2022.

[233] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[234] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[235] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.