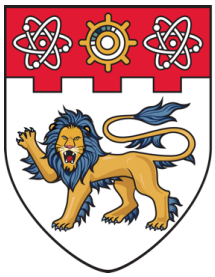


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Environment Poisoning in Reinforcement Learning: Attacks and Resilience

Hang Xu
School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Environment Poisoning in Reinforcement Learning: Attacks and Resilience

Hang Xu

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

23/09/2022

Date

Hang Xu

Hang Xu

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

23/09/2022

Date

Zinovi Rabinovich

Authorship Attribution Statement

This thesis contains materials from 2 papers published and 1 paper submitted in the following peer-reviewed conferences in which I am the first author.

Chapter 3 is published as **Hang Xu**, Rundong Wang, Lev Raizman, Zinovi Rabinovich, "Transferable environment poisoning: Training-time attack on reinforcement learning", Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (*AAMAS'21*), pp. 1398-1406, 2021.

The contributions of the co-authors are as follows:

- Asst/Prof Rabinovich provided the initial project direction.
- I designed the the study, performed all the laboratory work and analyzed the results. Rundong Wang provided suggestions on the experiment design and data analysis.
- I prepared the manuscript drafts. The manuscript was revised by Lev Raizman and Prof.Rabinovich.

Chapter 4 is published as **Hang Xu**, Xinghua Qu, Zinovi Rabinovich, "Spiking Pitch Black: Poisoning an Unknown Environment to Attack Unknown Reinforcement Learners", Proceedings of the 21th International Conference on Autonomous Agents and Multiagent Systems (*AAMAS'22*), pp. 1409-1417, 2022.

The contributions of the co-authors are as follows:

- Prof. Rabinovich provided instructive comments on the idea of this work.
- I designed the the study, performed all the laboratory work and analyzed the results.
- I prepared the manuscript drafts. The manuscript was revised by Xinghua Qu and Prof.Rabinovich.

Chapter 5 is published as **Hang Xu**, Xinghua Qu, Zinovi Rabinovich, "Reinforcement Learning Policy Resilience to Environment-Dynamics Poisoning", Thirty-sixth Conference on Neural Information Processing Systems (*NeurIPS'22*), Machine Learning Safety Workshop, 2022.

The contributions of the co-authors are as follows:

- Prof. Rabinovich provided instructive comments on the idea of this work.

- I designed the the study, performed all the laboratory work and analyzed the results.
- I prepared the manuscript drafts. The manuscript was revised by Xinghua Qu and Prof.Rabinovich.

23/09/2022

Hang Xu

Date

Hang Xu

Acknowledgements

Upon finishing my Ph.D. thesis, gratitude is the strongest feeling inside of me. I would like to express my sincere appreciation to all those who have provided me with invaluable help in this process.

I wish to express my greatest gratitude to my supervisor, Prof. Zinovi Rabinovich, not only for his expert guidance and scholarly advice on my research work, but also for his strong support and encouragement throughout my Ph.D. journey. His insightful comments and critical thinking always inspire me in tackling research problems. His attitude towards the academic research provides me with a good model of what a true researcher should be. Without his patient instruction and constant encouragement, I could not fulfil these research works and complete the Ph.D. study.

I would like to thank all talented members in Prof. Rabinovich's team, including Rundong Wang, Ridhima Bector and Wei Qiu, for the kind help and support that made my study in NTU a wonderful time. Besides, I would like to thank my collaborators, Xinghua Qu and Lev Raizman, for their valuable feedback and precious suggestions on my research studies. Many thanks to my friends, Yanling Li and Shuyang Ding, for the joy and the encouragement they gave me.

I would like to thank my Thesis Advisory Committee (TAC) members, Prof. Sinno Jialin Pan and Dr. Fedor Duzhin, for their insightful comments and suggestions on my research work. I also want to thank Mr. Kesavan Asaithambi for his support in Computational Intelligence Lab.

Most importantly, I would like to thank my parents, Zhaoxin Xu and Jiaojun Han, for their unconditional love and support throughout my life. I would like to express special thanks to my husband and best friend, Hantao Huang, for always being there for me and supporting me at all times. Having you in my life makes me feel incredibly lucky every single day. This thesis is dedicated to all of you.

Abstract

As Reinforcement Learning (RL) systems have been widely adopted in real-world applications, the security of these systems has become increasingly significant. Thus, it is essential to protect RL systems against a variety of adversarial attacks. Of these attacks, a training-time attack is considered to be particularly insidious as it poisons an RL policy itself. Training-time attacks attempt to force an RL agent to learn an attacker-desired policy by manipulating the agent's interaction information, such as reward signals and environmental responses. However, to achieve success with training-time attacks, it is generally considered that significant access to the RL system is necessary, such as the ability to distort the RL agent's reward values or experience memory. Also, it is assumed that the attacker possesses a comprehensive knowledge of the RL agent's learning algorithms, policy models, and/or its environment models. The assumption of such an omnipotent attacker makes a training-time attack unrealistic, and thus poses limited threat to an RL system in the real world. In contrast, this thesis studies training-time attacks with limited prior knowledge of RL systems, assuming only the ability to alter the training environment hyper-parameters (i.e., causal factors in physical systems) which are most likely to be accessed by third parties.

In this thesis, we investigate the security threats posed by training environment hyper-parameters and present a solution to address their adverse impact on RL policies. This thesis consists of three parts, including environment poisoning attacks (EPA) with full or limited prior knowledge, as well as a policy-resilience mechanism against the proposed attacks.

White-Box Environment Poisoning Attack: We propose a transferable environment poisoning attack (TEPA) against RL at training time, assuming that the attacker has full knowledge of an RL agent's learning mechanism (i.e., policy training algorithm and policy model structure/parameters) and its environment model (i.e., dynamics functions). We formulate the attack framework as a bi-level Markov Decision Process, seeking adaptive and minimal environment changes that

will prompt the momentary policy of the RL agent to change in an attacker-desired manner. Additionally, we demonstrate the transferability of our attack strategy. Specifically, an attack strategy, which is learned on a proxy agent, can be transferred to poison other victim agents' policies in the same tasks even if these victims adopt different learning algorithms and policy models. When compared to the existing reward-poisoning attacks, experimental results show that TEPA succeeds in efficiently forcing an RL victim to learn an attacker-desired policy via minimized changes to the training environment. Additionally, TEPA is empirically effective in poisoning a black-box RL agent as well as a population of RL agents due to its transferability property.

Double-Black-Box Environment Poisoning Attack: We propose a Double-Black-Box Environment Poisoning Attack (DBB-EPA) which requires minimal prior knowledge of the RL system. DBB-EPA assumes only the capability to alter the environment hyper-parameters and seeks to achieve policy compulsion on a *black-box* RL agent in a *black-box* training environment. To this end, we first investigate how to infer the internal information of an RL system and then learn an adaptive attack strategy based on the approximation of our attack objective. Empirical studies have demonstrated that DBB-EPA is effective against both tabular-RL and deep-RL agents in discrete as well as continuous state domains. We conclude that DBB-EPA poses threats more realistically to complex RL systems than TEPA which is restricted to white-box discrete environments.

Policy Resilience to Environment Poisoning Attack: To address the adverse effects of environment-poisoning attacks (i.e., TEPA and DBB-EPA) on an RL agent's policy learning, we propose a policy-resilience mechanism that attempts to recover poisoned policies in order to achieve optimal deployment performances. The policy-resilience mechanism is designed as a federated architecture in conjunction with a meta-learning manner, allowing efficient extraction and sharing of critical environment-structure knowledge. By leveraging the shared environment knowledge, agents can quickly grasp the dynamic features of deployment environments and accordingly generate imagined trajectories for recovering their poisoned policies. Such a policy-resilience procedure is summarized as three stages, namely, preparation, diagnosis and recovery. Empirical results show that the policy-resilience mechanism against TEPA and DBB-EPA is effective and efficient at recovering poisoned policies based on shared knowledge of the environment.

In summary, this thesis studies training environment poisoning attacks against RL in white-box and black-box settings, respectively. Such adversarial attacks reveal the vulnerability of RL to maliciously manipulated training environments. To protect the performance of RL policies, this thesis further develops a policy-resilience mechanism against the proposed attacks. Hopefully, this thesis serves as a starting point for investigating the security threats posed by training environments to RL policies, which could pave the way for the development of secure RL-based applications.

Keywords: Reinforcement Learning, Environment Poisoning, Policy Resilience

Contents

List of Figures

List of Tables

Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
DBB-EPA	Double-Black-Box Environment Poisoning Attack
DDPG	Deep Deterministic Policy Gradient
DP	Dynamics Programming
DQN	Deep Q-Network
DDQN	Dueling Deep Q-Network
DRL	Deep Reinforcement Learning
EPA	Environment Poisoning Attack
FGSM	Fast Gradient Sign Method
FL	Federated Learning
HiP-MDP	Hidden Parameter Markov Decision Process
IoT	Internet of Things
KLR	Kullback-Leibler Divergence Rate
LSTM	Long Short Term Memory
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
MSE	Mean Square Error
MVE	Model-based Value Expansion
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SOTA	State-of-the-Art
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient

TEPA	Transferable Environment Poisoning Attack
TRPO	Trust Region Policy Optimization
VPG	Vanilla Policy Gradient

Chapter 1

Introduction

1.1 Background and Motivations

The security of Reinforcement Learning (RL) has become increasingly significant due to the widespread deployment of RL systems in safety-critical applications, such as autonomous cars[1–3], smart energy systems[4–6] and healthcare systems[7, 8]. To protect RL applications from security threats, it is necessary to identify and analyse vulnerabilities of RL algorithms. Therefore, the study of RL's susceptibility to adversarial attacks has become a prominent research area of its own.

Existing adversarial attacks against RL can be classified into two categories: test-time attacks and training-time attacks. Specifically, the majority of existing works [9–13] focus on test-time attacks, challenging an already learned and fixed RL policy. These works successfully attack and degrade the performance of a deployed, RL-generated policy, though the policy itself is not subject to any manipulation. More recently, training-time attacks [14–18] have shown intensifying concern about security threats to the learning of RL policies. For example, there exists the possibility that the chatbot can be mistaught by a small group of Twitter users to make misogynistic and racist comments [19]; it is possible for autonomous cars to misread traffic signs if the signs are contaminated by adversarial stickers [20]; it has been observed that recommendation systems can be tricked by fake clicks and comments in order to rank products higher than they ought to be. In these examples, the policies of the RL agents have been poisoned, which is a serious security

issue in RL-based applications. Therefore, the purpose of this thesis is to examine the security threats associated with the learning of RL policies.

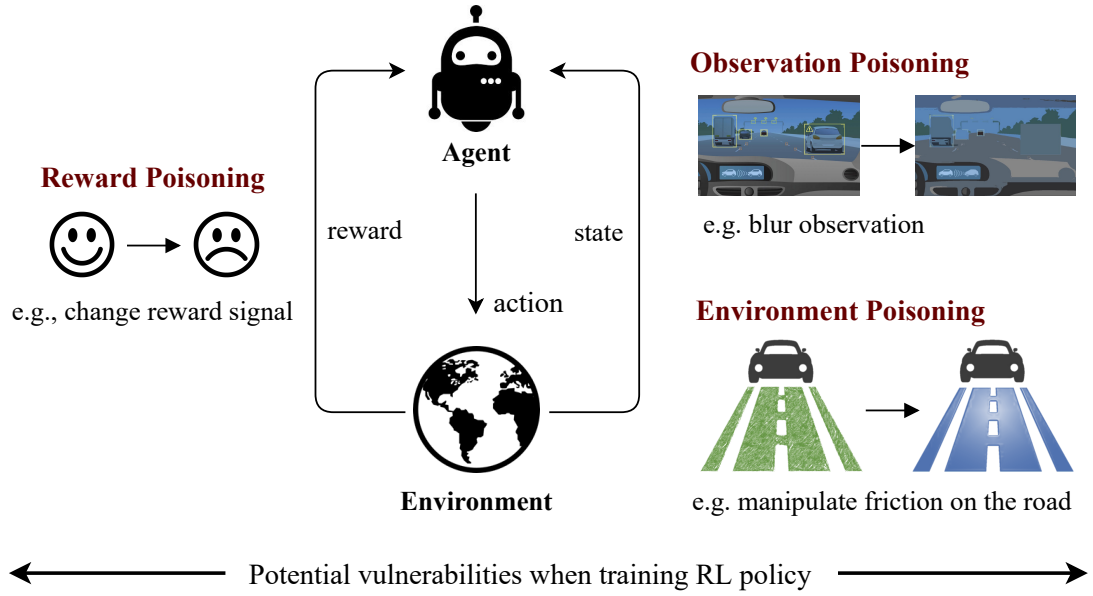


FIGURE 1.1: Potential vulnerabilities during the training of RL policies, such as reward signals, perceived states (i.e., observations) and environment dynamics.

As shown in Figure 1.1, existing training-time attacks perturb the feedback (i.e., environmental responses) that an agent perceives from its environment during the training time. These attacks succeed in poisoning the agent's policy, assuming a significant access to the agent's perception, e.g., perceived rewards and perceived states. Such an assumption is difficult to be satisfied, which limits the threats of these attack approaches. In contrast, this thesis studies the security threats caused by *environment hyper-parameters*, aiming to poison RL policies without access to the agent's perceptions or memories. Here, it is important to distinguish the environment hyper-parameters with the RL-algorithm hyper-parameters (e.g., exploration rate or learning rate). As shown in Figure 1.2, environment hyper-parameters are also termed causal factors in physical systems [21–23], such as the friction of the surface, the resistance in the air and the slope of the mountain. These hyper-parameters can not be observed directly, but their effect can be 'felt' by the RL agent when interacting with the environment. This means that environment hyper-parameters affect how the environment responds to the RL agent's actions. Namely, they can be used to parameterize the environment dynamics [21, 22, 24] so that altering hyper-parameters leads to changes of the environment dynamics. As a result, the manipulated training environment could poison the learning of RL

agent's policy, resulting in attacker-desired or non-optimal behaviours during the agent's deployment. Therefore, it is imperative to investigate the threats posed by environment hyper-parameters as well as to develop protection solutions for RL policies.

FIGURE 1.2: Examples of environment hyper-parameters.

Furthermore, the success of existing training-time attacks [14–16] relies on comprehensive prior knowledge of the RL system, including RL agent's learning mechanism (i.e., learning algorithm and policy model) and/or its environment model (i.e., transition dynamics and reward functions). Unfortunately, assuming such an omniscient attacker makes attack approaches somewhat unrealistic, which limits threats to real-world RL-based applications. This thesis attempts to address such a limitation by studying a novel training-time attack which requires minimal prior knowledge of an RL system.

1.2 Major Contributions

This thesis investigates the security problem of RL policy learning from two perspectives: adversarial attacks and protection solutions. Specifically, we propose training environment-poisoning attacks (EPA) against RL based on different assumptions about the attacker's prior knowledge and access ability, followed by a study of policy-resilience mechanism. A summary of the major contributions is presented as Figure 1.3.

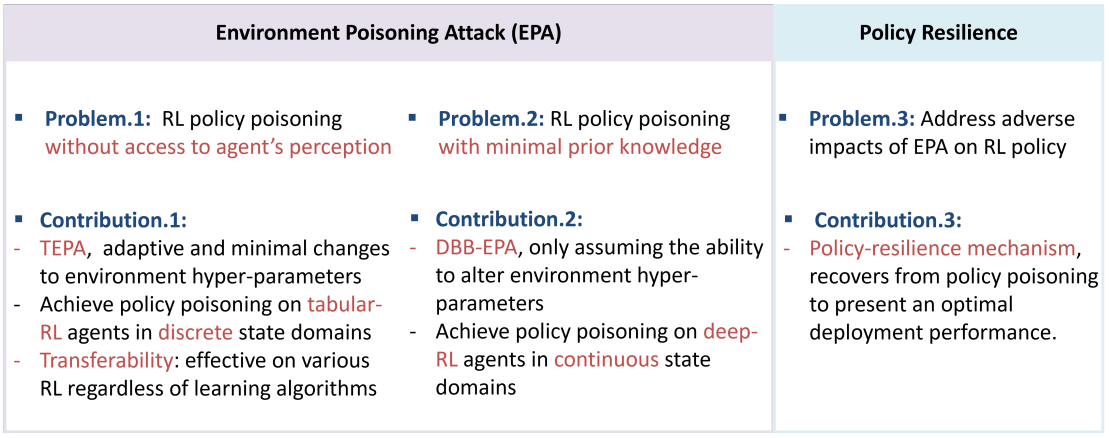


FIGURE 1.3: Summary of research problems and corresponding contributions.

White-Box Environment Poisoning Attack: We propose a transferable environment poisoning attack (TEPA) against RL at training time, with an assumption that the RL agent's learning mechanism (i.e., policy training algorithm and policy model structure/parameters) and its environment-dynamics model are known by the attacker. We design an attack framework as a bi-level Markov Decision Process for learning an adaptive attack strategy which can force an RL agent to learn an attacker-desired policy via minimized environment changes. Moreover, the attack strategy is demonstrated to possess transferability property. Namely, an attack strategy, which is learned based on one RL agent (i.e., proxy agent), can be transferred to attack other RL agents which learn the same task but adopt different algorithms or policy structures. Relying on this property, we propose that TEPA is effective in poisoning various RL agents' policies even if these agents are black-box (i.e., unknown learning algorithms and policy model). Furthermore, a variation of TEPA is developed for a population of independent and isolated RL agents with the goal of manipulating behaviour across multiple agents via minimized cumulative changes in their environments. Finally, we empirically evaluate TEPA against a tabular RL agent in discrete state domains and show that it has superior effectiveness and efficiency in comparison with classical reward poisoning attacks.

Double-Black-Box Environment Poisoning Attack: We propose a Double-Black-Box Environment Poisoning Attack (DBB-EPA) with minimal assumptions on the attacker's prior knowledge, namely both the RL agent and the training environment are black-box to the attacker. Assuming only the ability to alter the

environment hyper-parameters, DBB-EPA aims to achieve policy compulsion with minimal and adaptive environment poisoning. To achieve this goal, we investigate how to infer the internal information of an RL system, and then we design an approximation of our attack objective based on the inferred information. Specifically, given observations of an RL agent's trajectories during its learning process, we jointly train: 1) an Encoder-Dual-Decoder network that learns a low-dimensional latent representation of the RL system's internal information; 2) an attack strategy, conditioned on the latent representation and environment hyper-parameters, that manipulates the RL agent's policy using minimal environment changes. According to experimental results, DBB-EPA poisons the policies of both tabular-RL and deep-RL agents in discrete as well as continuous state domains. As opposed to TEPA which is limited to white-box environments, DBB-EPA provides a more realistic way to pose threats to complex RL systems in which environment models are unknown.

Policy Resilience to Environment Poisoning Attack: We propose a policy-resilience mechanism against EPAs in order to recover an agent's poisoned policy for an optimal deployment performance. This policy-resilience procedure can be described as a three-step process that includes preparation, diagnosis, and recovery. Specifically, the knowledge of the environment structure is extracted at the preparation stage and then shared with a poisoned agent. Depending on the shared environment knowledge, the agent is able to quickly and accurately comprehend the dynamics of deployment environments at the diagnosis stage. At the final stage, the agent recovers its poisoned policy based on imagined trajectories derived from the dynamics model of the deployment environment. As a key idea of the policy-resilience mechanism, extracting and sharing environmental knowledge is achieved by the design of a federated architecture coupled with a meta-learning manner. Evaluation of the policy-resilience mechanism against TEPA and DBB-EPA demonstrates its effectiveness and efficiency in restoring policy deployment performance. With regard to the idea of sharing knowledge, this proposed policy-resilience mechanism represents a resource-efficient and time-conserving method of addressing harmful effects caused by environment-poisoning attacks.

1.3 Thesis Outline

The rest of this thesis is organized as follows.

Chapter 2 provides preliminary knowledge about RL and reviews literature of adversarial attacks and defense methods, followed by a discussion about our contributions to the research community. Chapter 3 designs a transferable environment-poisoning attack with full knowledge of the RL system, and presents the transferability of the attack strategy. Chapter 4 introduces a double-black-box environment-poisoning attack with minimal assumptions on the attacker's prior knowledge, providing a more realistic attack approach. Chapter 5 presents a policy-resilience mechanism against the proposed environment-poisoning attacks. Chapter 6 summarizes this thesis and describes future works.

Chapter 2

Preliminaries and Literature Review

In this chapter, we first introduce RL backgrounds and then present existing works about adversarial attacks on RL, followed by defense approaches against these attacks. We further discuss our contributions with regards to existing literature.

2.1 Reinforcement Learning

In this section, we present an overview of reinforcement learning and briefly describe tabular-RL methods and deep-RL algorithms which are involved in this thesis.

2.1.1 Overview

As shown in Figure 2.1, the foundation of Reinforcement Learning (RL) lies in the idea of learning through interaction with the environment. At each time step the agent (i.e., learner) interacts with the environment, the agent decides what action to take based on its observation of a state. The environment responses to the action and present a new state to the agent. Also, the agent receives a reward signal from the environment, in the form of a number that indicates the quality of the agent's choice of actions. Accordingly, rewarding or punishing the agent for their behaviour makes him or her more likely to repeat that behaviour or abandon

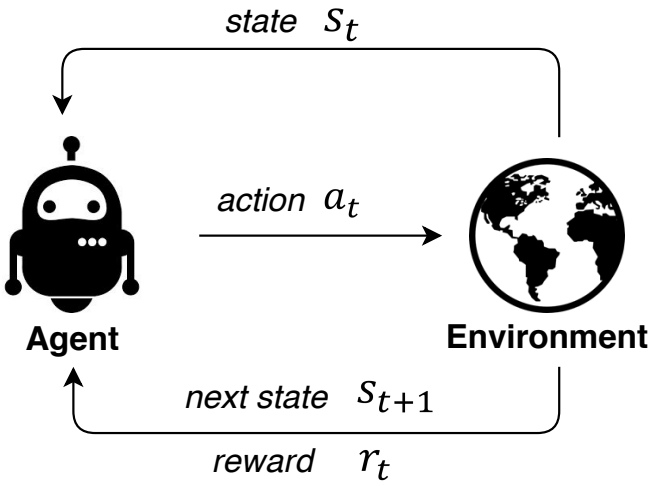


FIGURE 2.1: Illustration of reinforcement learning.

it in the future. The agent is attempting to learn a behaviour that maximizes cumulative rewards for achieving its tasks. Thus, such *trial-and-error* is one of the most distinguishing features of RL.

Formulation. Specifically, the problem of reinforcement learning is formalized as the optimal control of Markov Decision Process (MDP). An MDP is formulated as the 6-tuple $M = \langle S, A, T, R, d_0, \gamma \rangle$, where

- $s \in S$ is a representation of the environment state;
- $a \in A$ is one action of the agent's action space;
- $T : s \times a \to \mathcal{R}$ is an environment dynamics function $T(s'|s, a)$ that tells the transition probability from s to s' given an action a;
- $R : s \times a \times s' \to \mathcal{R}$ is a reward function $R(r|s, a, s')$ that represents the reward when the agent takes the action a to transit from the state s to the state s';
- $d_0(s)$ is a distribution over the agent's initial states;
- γ is a discount factor.

A sequence of states and actions is termed as a trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$. The MDP provides an abstract and flexible framework that can be applied to a wide range of RL problems in various ways.

Objectives. An RL agent's goal is to learn a behaviour policy $\pi(a|s)$ that maximizes the cumulative rewards over a trajectory (i.e., Return G_t), denoted as

$$\max_{\pi} \sum_{t}^{\infty} \gamma^t R_t.$$

Value Functions. The policy learning is evaluated by value functions, including the state-value function and the action-value function.

A *state-value function* refers to the expected return when beginning at state s and following policy π thereafter, which is defined as follows

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s) \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{for all } s \in S.$$

An *action-value function* is defined as the expected return beginning at state s, taking the action a, and thereafter following policy π:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a) \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right].$$

Importantly, a fundamental property of value functions is the recursion relationship between the immediate reward of a state and the discounted values of its successor states. Thus, value functions can be expressed as *Bellman Expectation Equations*, denoted as

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s \right] \\ &= \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V_\pi(s') \right), \end{aligned}$$

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma Q_\pi(s_{t+1}, A_{t+1}) \mid s_t = s, A_t = a \right] \\ &= R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q_\pi(s',a'). \end{aligned}$$

Accordingly, an optimal policy π^* can be obtained by searching for an optimal state-value function V^* or an optimal action-value function Q^*, which satisfy the

Bellman optimality equations denoted as

$$V^*(s) = \max_\pi V_\pi(s) \text{ for all } s \in S;$$
$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \text{ for all } s \in S \text{ and } a \in A.$$

Almost all RL algorithms involve the estimation of value functions.

2.1.2 Tabular-RL Algorithms

Classical RL algorithms are mainly classified as three categories: Dynamics Programming (DP), Monte-Carlo (MC) Methods and Temporal-Difference (TD) Learning [25]. Here, DP is an algorithm capable of computing optimal policies based on environment models, including dynamic functions and reward functions, which is applicable to model-based RL problems. MC and TD are used for solving model-free RL problems where the environment model is unknown. In this part, we focus on the introduction of MC and TD as our thesis studies model-free RL problems.

Monte Carlo Methods. Monte Carlo (MC) methods solve an RL problem based on actual or simulated experience samples, i.e., sequences of states, actions and rewards from interactions with an environment. The value function of a given policy π can be approximated by averaging sample returns, which is denoted as

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}[G_t \mid s_t = s].$$

Since MC uses empirical mean returns for estimating value functions, it is suitable for solving model-free RL problems but only for episodic tasks.

Temporal-Difference Learning. Similar to MC methods, Temporal-Difference (TD) methods can learn an optimal policy based on raw experience without a model of the environment dynamics. However, TD methods estimate the value functions without waiting for the final outcome, but instead, using other estimated value

functions. Specifically, the simplest TD method $TD(0)$ estimates $V(S_t)$ immediately based on transition to S_{t+1} and receiving R_{t+1}, denoted as

$$V(a_t) \leftarrow V(s_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(s_t)\right].$$

Sarsa is an on-policy TD method, which estimates action-value function Q as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].$$

This update is done after every transition $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ where the next action a_{t+1} is sampled from the policy derived from Q.

Q-learning is an off-policy TD method, which directly approximate the optimal action-value function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)\right],$$

which is independent of the policy being followed. Note that the difference between *Sarsa* and Q-learning is how to choose the action for the next state s' during the updates.

2.1.3 Deep-RL Algorithms

With the development of Deep-RL (DRL) algorithms, RL has successfully been used to create breakthrough AIs for sophisticated decision-making tasks, such as strategy games Go and Dota [26], simulated or real-world robots [27] and autonomous cars [2]. In this part, we will briefly introduce DRL algorithms which are involved in this thesis.

Deep Q-Network (DQN). DQN [28] is a deep-learning-based Q-learning algorithm, which approximates action-value function Q using a neural network (i.e., Q_θ network). Specifically, the Q_θ network takes a state-action pair as an input and outputs the prediction of expected cumulative rewards. The Q_θ network utilizes

stochastic gradient descent to minimize the following loss

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{(s,a,s',r)\sim\mathcal{B}} \left[\left(r + \max_{a'} \hat{Q}_{\theta^-}(s', a') - Q_\theta(s, a) \right)^2 \right].$$

Here, $\mathcal{B}$ is a replay buffer that stores transition data sampled by history policies and provides a batch of data to the Q network for iterative training. Also, $\hat{Q}_{\theta^-}$ is a target network which is cloned from the Q_θ network and kept frozen for a fixed number of episodes, providing the Q network with target value. Note that both the replay buffer and the target network contribute to the stabilized learning of an optimal policy. The details of DQN is described as pseudocode in Algorithm 1. Finally, with the optimized Q network, the policy is denoted as

$$\pi(s) = \arg\max_a Q_\pi(s, a)$$

Algorithm 1 Deep Q-network (DQN)

1: Initialize action-value function Q_θ and target network $\hat{Q}_{\theta^-}$, where $\theta^- = \theta$. Initialize a replay buffer $\mathcal{B}$.
2: **for** episode i= 1,2,... **do**
3: **for** time steps t=1,2,... **do**
4: Given a state s, choose a based on Q_θ (using ϵ-greedy)
5: Take action a, receive next state s' and reward r
6: Store transition (s, a, r, s') to replay buffer $\mathcal{B}$
7: Sample a minibatch of transitions $\{(s, a, r, s')_n\}_{n=1}^N$ from $\mathcal{B}$
8: Calculate $y_n = r_n + \max_{a'} \hat{Q}_{\theta^-}(s_n^{'}, a')$
9: Update weights θ by minimizing $(y_n - Q_\theta(s_n, a_n))^2$
10: Update target network $\theta^- = \theta$ every C steps
11: **end for**
12: **end for**

Vanilla Policy Gradient (VPG). Policy gradient algorithms [25] consider the case of stochastic, parameterized policy π_θ. The key idea behind policy gradients is to increase the probability of actions that result in higher returns, and to decrease the probability of actions that result in lower returns, until an optimal policy that maximizes the expected return as $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}[R(\tau)]$. Here, the policy is optimized using gradient ascent, which is denoted as $\theta_{k+1} \leftarrow \theta_k + \alpha\nabla_\theta J(\pi_\theta)$. Note that $\nabla_\theta J(\pi_\theta)$ is termed as policy gradient, which is should be numerically expressed for computation. Specifically, VPG is an on-policy algorithms, which expresses the

policy gradient as

$$\nabla_\theta J_{\pi_\theta} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} A^{\pi_\theta}(s_t^n, a_t^n)\nabla_\theta \log \pi_\theta(a_t^n|s_t^n).$$

where N is the number of trajectory τ and A^{π_θ} is the advantage estimate. A^{π_θ} can be estimated as using rewards-to-go $\hat{R} = \sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n$ and state-value function $V_\phi(s)$, denoted as $A^{\pi_\theta}(s,a) = \hat{R} - V_\phi(s)$. Note that the trajectory is obtained using Monte Carlo sampling, thus, the stochastic policy π is updated based on empirical returns at the end of each episode. The details of VPG is described as pseudocode in Algorithm 2.

Algorithm 2 Vanilla Policy Gradient (VPG)

Initialize policy parameters θ and value function parameter ϕ
for i= 1,2,... **do**
  Collect a set of trajectories $\mathcal{D}_i = \tau_n$ by running policy π_{θ_i} in the environment
  Compute rewards to go $\hat{R}_t$
  Compute advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_i}
  Estimate policy gradient as $\hat{g}_i = \frac{1}{|\mathcal{D}_i|}\sum_{\tau\in\mathcal{D}_i}\sum_{t=1}^{T_n} A^{\pi_{\theta_i}}(s_t,a_t)\nabla_\theta \log \pi_{\theta_i}(a_t|s_t)$
  Compute policy update $\theta_{i+1} = \theta_i + \alpha\, \hat{g}_i$
  Fit value function $\phi_{k+1} = \underset{\phi}{\operatorname{argmin}} \frac{1}{\|\mathcal{D}_i\|\times T}\sum_{\tau\in\mathcal{D}_i}\sum_{t=1}^{T_n}\left(V_{\phi_k}(s_t) - \hat{R}_t\right)^2$
end for

Proximal Policy Optimization (PPO). PPO [29] is a variance of policy gradient, which enables multiple epochs of minibatch updates while VPG performs only one gradient update per data sample. As a result, the sample efficiency and learning speed are improved. Specifically, when optimizing policy π using PPO, an older policy $\pi_{\theta'}$ is used to interacts with environment for generating trajectory samples. To offset the probability sampling from a different distributions $\pi_{\theta'}$, the idea of importance sampling is adopted, i.e., a sampling weight $\frac{\pi}{\pi_{\theta'}}$ is required. In addition, the deviation between $\pi_{\theta'}$ and π_θ should be constrained to achieve accurate importance sampling. Based on the constraints, there are two kinds of PPO algorithms: PPO-Penalty and PPO-Clip.

First, PPO-Penalty penalizes the Kullback-Leibler divergence $KL(\theta,\theta')$ in the objective function, denoted as

$$J^{\theta'}_{Penalty}(\theta) = \mathop{\mathbb{E}}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right] - \beta KL(\theta,\theta'),$$

where β is the penalty coefficient which can be automatically adjusts.

Second, PPO-Clip utilizes specialized clipping in the objective function to prevent the new policy from diverging far from the old one. The objective function is denoted as

$$J^{\theta'}_{Clip}(\theta) = \sum_{(s_t,a_t)} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t),\ \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A^{\theta'}(s_t,a_t)\right),$$

where ϵ is a hyper-parameters which is generally set as 0.1 or 0.2.

The pseudocode of PPO is very similar to VPG (i.e., Algorithm 2), except the definition of the objective function and the sampling method.

Deep Deterministic Policy Gradient (DDPG). DDPG [30] is an RL algorithm used for environment with continuous action spaces. It concurrently learns a Q function and a policy, which are represented by a Q network (i.e., critic network) and a policy network (i.e., actor network), respectively.

Specifically, DDPG uses off-policy data and the Bellman equation to learn an approximator to the Q-function $Q^*(s,a)$, and uses the Q-function to learn an approximator to the deterministic policy $\mu^*(s)$. It shares the same tricks of DQN that adopts replay buffer and target network to stabilize the learning of Q network and policy network. Thus, the Q network Q_ϕ is optimized by minimizing the following loss function as

$$\mathcal{L}(\phi,\mathcal{B}) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma \max_{a'} Q_{\bar{\phi}}(s',a')\right)\right)^2\right],$$

where $Q_{\bar{\phi}}$ is the target Q network and $\mathcal{B}$ is the replay buffer. Note that a' is generated by the target policy network $\mu_{\bar{\theta}}(s')$.

The deterministic policy $\mu_\theta(s)$, which gives the action that maximizes $Q_\phi(s,a)$, is optimized using gradient ascent (with respect to policy parameters only) to solve

$$\max_\theta \mathop{\mathbb{E}}_{s\sim\mathcal{B}}\left[Q_\phi(s,\mu_\theta(s))\right].$$

Furthermore, to stabilize and improve the DDPG learning, a Twin Delayed DDPG (TD3) [31] is proposed which involves three tricks: clipped double Q-learning,

delayed policy updates and target policy smoothing. The details of DDPG is described as pseudocode in Algorithm 3.

Algorithm 3 Deep Deterministic Policy Gradient (DDPG)

Initialize critic network Q_ϕ and target critic network $\hat{Q}_{\phi^-}$, where $\phi^- = \phi$. Initialize actor network μ_θ and target actor network μ_θ, where $\theta^- = \theta$. Initialize a replay buffer $\mathcal{B}$.
for episode i= 1,2,... **do**
    for time steps t=1,2,... **do**
        Given a state s, choose a using μ_θ with noisy
        Take action a, receive next state s' and reward r
        Store transition (s, a, r, s') to replay buffer $\mathcal{B}$
        Sample a minibatch of transitions $\{(s, a, r, s')_n\}_{n=1}^N$ from $\mathcal{B}$
        Calculate $y_n = r_n + \gamma \hat{Q}_{\phi^-}(s_n^{'}, \hat{\mu}_{\theta^-}(s_n^{'}))$
        Update critic network by minimizing $\frac{1}{N}\sum_n^N \left(y_n - Q_\theta(s_n, a_n)\right)^2$
        Update actor network via gradient $\frac{1}{N}\sum_n^N \nabla_a Q_\phi(s_n, a)_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s_n)$
        Update target networks $\phi^- = \tau\phi + (1-\tau)\phi^-$, $\theta^- = \tau\theta + (1-\tau)\theta^-$
    end for
end for

2.2 Adversarial Attacks against RL

We begin this section by providing an overview of adversarial attacks in machine learning. Following that, we review existing works about attacks against reinforcement learning, including test-time attacks and training-time attacks.

2.2.1 Overview of Adversarial Attacks

Machine Learning (ML) algorithms consist of supervised learning, unsupervised learning and reinforcement learning. With the rapid development of ML techniques, security problem has been introduced as a new challenging in designing and developing robust intelligent systems. Existing works on adversarial attacks of ML can be classified from perspectives, namely, attacker's target, attacker's knowledge and attacker's goal.

Attacks based on attack target There are mainly two categories of security attacks in terms of the attack target:

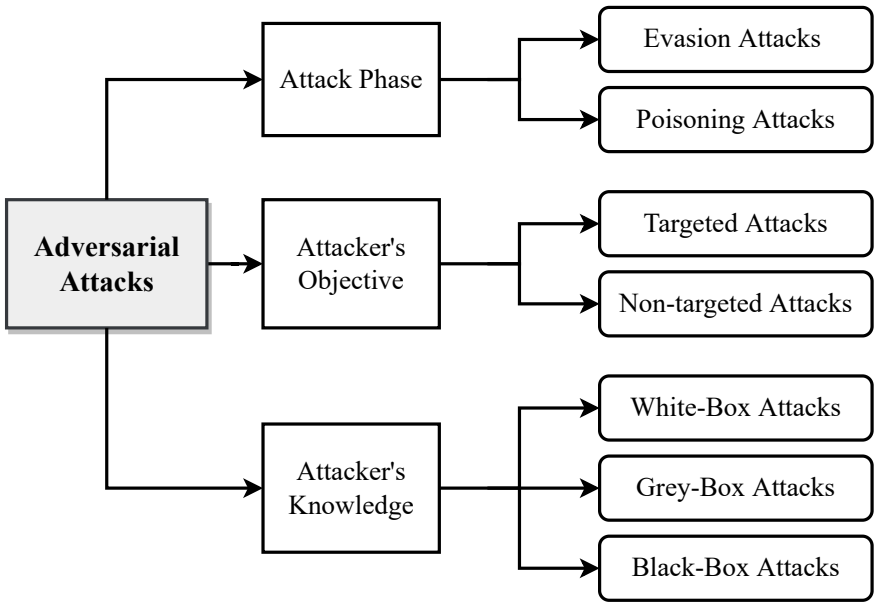


FIGURE 2.2: Categories of adversarial attacks against machine learning.

- **Evasion attacks** provide malicious inputs to the trained model/policy and aim to mislead the model/policy to provide faulty results/actions [32]. Here, the crafted malicious input is termed as an adversarial example. The evasion attacks is also termed as the inference-phase attacks or test-time attacks.
- **Poisoning attacks** poison training data by introducing imperceptible perturbations to the training dataset [33] to force the system to learn a defective model/policy. Thus, the poisoning attack is also termed as training-time attacks.

Attacks based on attacker's knowledge: In terms of the knowledge of the attacker, there are three types of attacks: white-box attacks, grey-box attacks and black-box attacks.

- **White-box attacks** assume the attacker has full knowledge of the targeted ML algorithm. The attacker knows the training data, learning algorithm as well as the parameters of the model.
- **Grey-box attacks** assume the attacker has limited access to the targeted ML model, i.e., it knows either learning algorithm or model representations.

- **Black-Box attacks** assume the attacker has no information about the targeted ML model. It is only able to ask for labels or confidence from the intelligent systems, to accordingly construct adversarial examples.

There is a key property, *transferability*, of adversarial examples [34], which enables the development of gray-box and black-box attacks. The term *transferability* indicates that adversarial examples, which can mislead one model, may also disturb another one. It has been demonstrated effective for models whose architectures or training sets are different, given that these models are trained for the same task. Due to the transferability property, attackers can use proxy ML models to develop attack strategies that can then be applied to ML targets.

Attacks based on attacker's goal: From the perspective of the attacker's objectives, there are mainly two categories of adversarial attacks, taking classification tasks as an example:

- **Non-targeted Attacks** where the attacker aims to cause misclassification via perturbing the classification boundary of any class;
- **Targeted Attacks** where the attacker aims at misleading the deployed classifier to a specific class.

In recent years, the security issue related to RL has become increasingly. The following sections will present existing adversarial attacks against RL from two perspectives: test-time attacks (i.e., evasion attacks) and training-phase attacks (i.e., poisoning attacks). These attack approaches are categorized and listed in Figure 2.3.

2.2.2 Testing-time Attacks

The majority of existing works focus on test-time attacks against RL. As illustrated in Figure 2.4, these attacks manage to mislead the agent's action decision so that degrade the deployment performance of the RL policy. It is important to note that the learned policy itself has not been poisoned by test-time attacks. One category of test-time attacks [9, 10, 35, 36] is developed by extending the idea

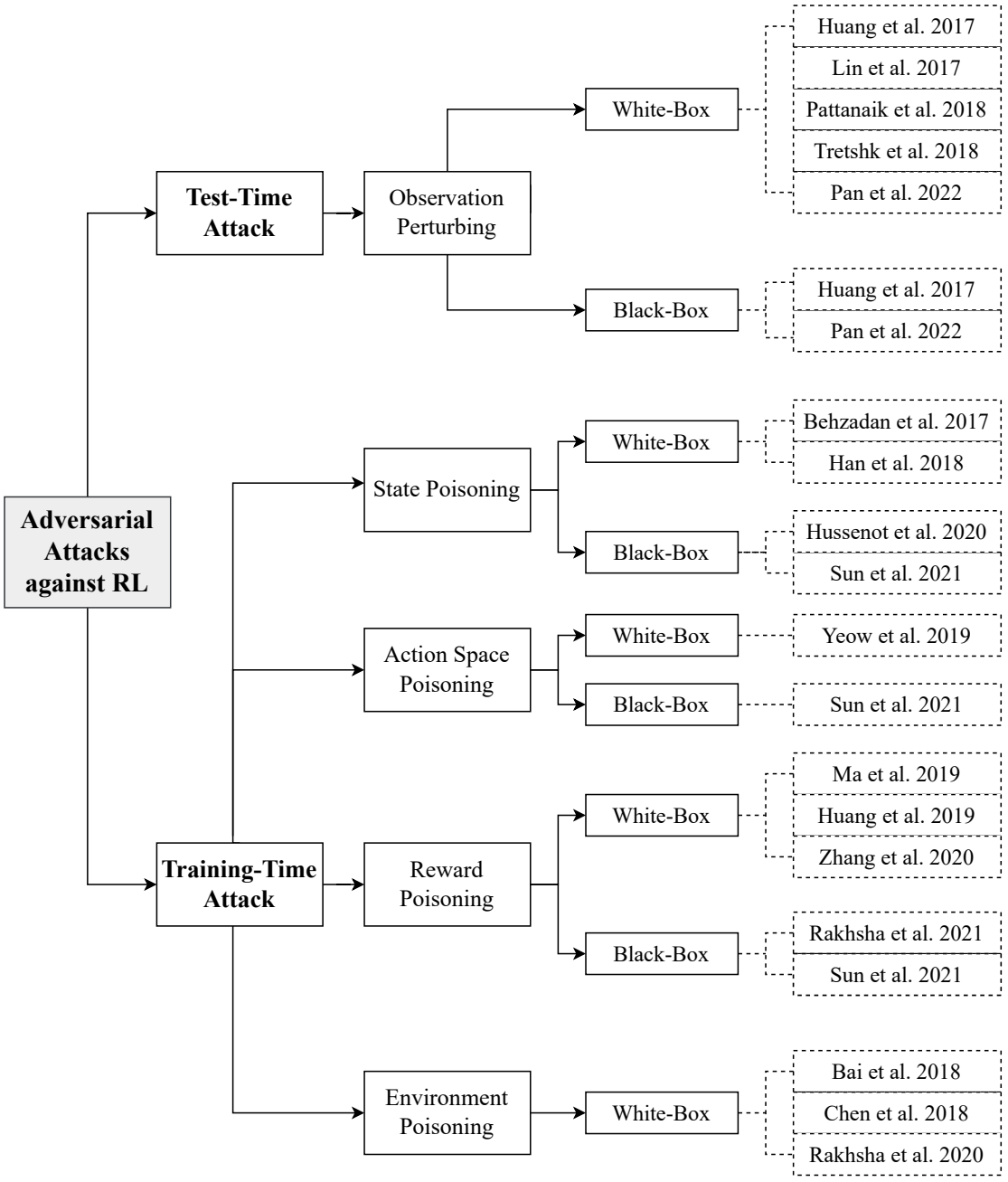


FIGURE 2.3: Literature reviews of adversarial attacks against RL.

of Fast Gradient Sign Method (FGSM) [37] to attack DRL agent's policy; while others [12] are investigated based on Adversarial Transformer Network [38]. Next, we describe some famous test-time attack approaches on DRL in chronological order as shown in Table 2.1

Huang et al. 2017 [9] first propose adversarial attacks that are effective on deep reinforcement learning (DRL). This work perturbs the agent's observations (i.e., raw input of the policy neural network) at every time step by adopting FGSM

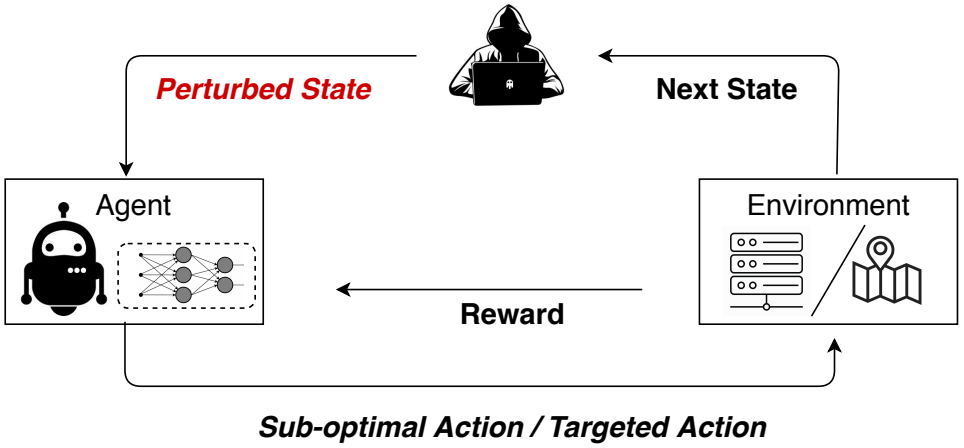


FIGURE 2.4: Illustration of test-time attack against RL.

TABLE 2.1: Summary of selected test-time attacks against RL.

Papers	Test-Time Attack		Attack Target		Targeted RL
	white-box	black-box	state	action	
Huang et al. 2017 [9]	✓	✓	✓		DQN, A3C, TRPO
Lin et al. 2017 [10]	✓		✓		DQN, A3C
Pattanaik et al. 2018 [35]	✓		✓		DQN, DDPG
Tretschk et al. 2018 [12]	✓		✓		DQN
Pan et al. 2022 [36]	✓	✓	✓	✓	DQN, DDPG

[37], which causes a significant drop in test-time performance of the deployed DRL policies. Furthermore, this work investigates the vulnerability of various DRL algorithms in white-box and black-box settings. It concludes that policy trained with DQN is more vulnerable to adversarial attacks in comparison with that of trust region policy optimization (TRPO) [39] and asynchronous advantage actor-critic (A3C) [40]. This work represents the first attempt to determine whether RL systems are vulnerable to adversarial attacks, which motivates a series of new studies into the issue of adversarial attacks within the domain of RL.

Lin et al. 2017 [10] proposed two tactics, *strategically-timed attack* and *enchanting attack*, to attack DRL agents using adversarial examples. Different from the uniform attack in work [9] where the DRL agent is attacked at each time step in an episode, the proposed strategically-timed attack uses adversarial examples at selected time steps when the attack is effective. It aims at reducing the DRL agent's rewards with fewer crafted adversarial examples. Besides this non-targeted strategically-timed attack, an enchanting attack is proposed that maliciously leads the agent to a targeted state. As the first planning-based adversarial attack on

DRL, the enchanting attack utilizes a deep generative model to forecast the next state and a sophisticated planning algorithm to generate an action required to lead the agent there. Compared to previous FSGM-based attacks [9], the proposed attacks are more practically feasible and hard to be detected. They are evaluated on attacking DRL algorithm (i.e., DQN and A3C) in five Atari games and achieve 70% success rate.

Pattanaik et al. 2018 [35] improves gradient-based adversarial attacks to further reduce the DRL agent's return via perturbing its observations. The authors argue that, in the previous FSGM-based attack [9], the generated noise is not guaranteed to cause adversarial attack. To solve this limitation, they propose a loss function which maximizes the probability of identifying the worst possible action. They adopt the Stochastic Gradient Descent approach to generate adversarial action which maliciously lures the agent to terminate at a pre-defined state. When attacking DDQN agents in MountainCar and CartPole, such gradient-based attack approaches have better performance than the simple FGSM-based attack [9]. Compared to prior works [9, 10], the proposed attack is not constrained in attacking RL environment where input is of the high-dimensional pixel (i.e., images).

Instead of adopting gradient computation [9, 10, 35], Tretschk et al. 2018 [12] utilizes a feed-forward deep neural network, Adversarial transformer network [38], to generate perturbations which are added to the DRL agent's input states. This work imposes a targeted adversarial reward on the victim DRL network, i.e., the victim DRL agent is misguided to pursue an adversarial reward at test time, via a sequence of adversarial examples. Compared with the enchanting attacks [10] targeting at adversarial states, this work is more flexible due to that the adversarial reward is able to encode the adversarial state. This attack method is developed in white-box settings and evaluated on DQN trained for playing Pong.

A recent study conducted by Pan et al. 2022 [36] suggested that FGSM-based attacks in [9, 10] perturb the observation independently on each frame, thus demanding intensive computation during real-time implementation. As a solution to this problem, the authors propose online sequential attacks on the state of a DRL agent using its temporal consistency. In the proposed attack, a perturbation is generated based on the information in a few frames, and the perturbation is applied to subsequent frames. Furthermore, this work also investigates attacks on DRL agent's action space and environment dynamics. These proposed attacks

are evaluated on attacking DQN and DDPG agents in MuJoCo games and driving simulation TORCS. This online sequential attack is faster than the FGSM-based attacks [9, 10] due to that no back-propagation is required.

2.2.3 Training-time Attacks

Training-time attacks are different from test-time attacks in that they aim to alter an RL policy itself, as opposed to degrading the deployment performance of the policy. As shown in Figure 2.5, the attacker intervenes the policy learning by maliciously manipulating the agent's perceived states, selected actions, received rewards or training-environment dynamics. In this section, we provide descriptions of existing training-time attacks that can be classified as state poisoning (see Section 2.2.3.1), action space poisoning (see Section 2.2.3.2), reward poisoning (see Section 2.2.3.3) and environment poisoning (see Section 2.2.3.4). Table 2.2 summarizes existing training-time attack approaches which are presented in this section.

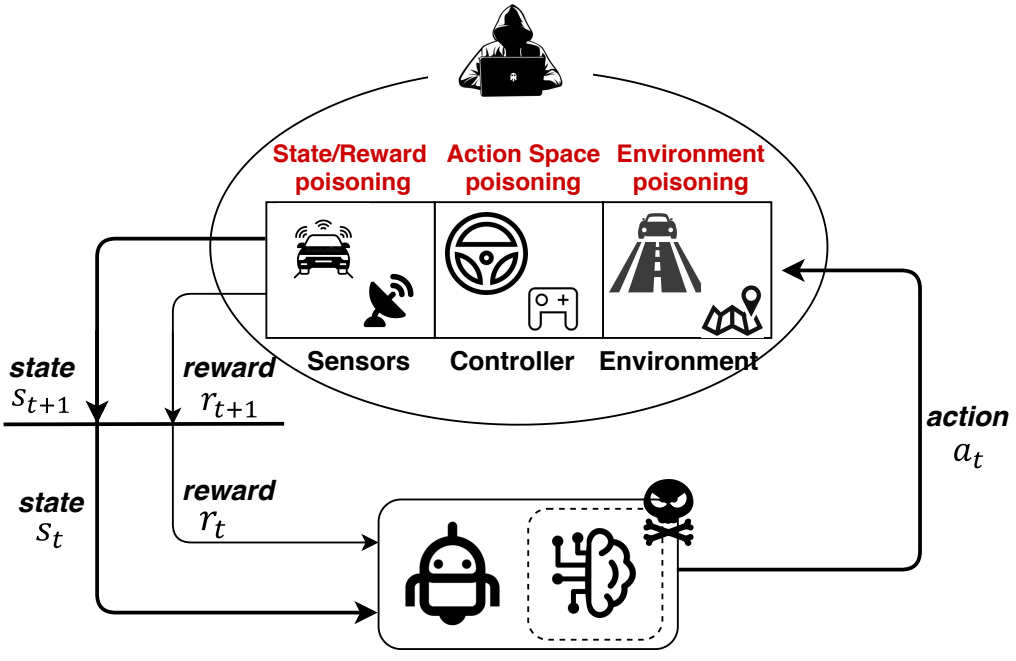


FIGURE 2.5: Illustration of training-time attacks against RL.

TABLE 2.2: Summary of training-time attacks against RL.

Papers	Training-Time Attack		Attack Target				Targeted RL
	white-box	black-box	state	action	reward	env.	
Behazadan et al. 2017 [41]	✓		✓				DQN
Han et al. 2018 [42]	✓	✓	✓				DDQN, A3C
Hussenot et al. 2020 [43]	✓	✓	✓				DQN
Sun et al. 2020 [18]		✓	✓	✓	✓		VPG, PPO, A2C
Yeow et al. 2019 [44]	✓			✓			PPO, DDQN
Ma et al. 2019 [14]	✓				✓		TCE, LQR
Huang et al. 2019 [45]	✓				✓		Q-learning
Zhang et al. 2020 [15]	✓				✓		Q-learning
Rakhsha et al. [17]		✓			✓		UCRL2
Bai et al. 2018 [46]	✓					✓	DQN
Chen et al. 2018 [47]	✓					✓	A3C
Rakhsha et al. 2020 [16]	✓					✓	DP, UCRL

2.2.3.1 State/Observation Poisoning

Behazadan et al. 2017 [41] propose the Policy Induction Attacks applied on RL during training time, which provides adversarial rewards to the agent for learning policy. The authors argue that DQN is susceptible to adversarial input perturbations (i.e., changes in the observable environment), and demonstrate the adversarial examples can be transferable across different DQN systems. The attacker in this study is assumed to have no prior knowledge of the DQN architecture. It stands between the victim agent and the environment, perturbs the configuration of the environment, and then provides poisoned states to the victim. The attacker attempts to lead the victim to undertake a series of actions that will result in adversarial rewards. In addition, the authors describe an attack approach that exploits the transferability of adversarial examples in order to manipulate policy. A 70% success rate has been achieved in transferring adversarial examples from one DQN model to another in black-box settings.

Han et al. 2018 [42] investigate how DRL agents react to adversarial attacks that poison the training process in Software-Defined Networks. The authors adopt DDQN and A3C as victim DRL agents, and study untargeted/targeted attacks in both white-box and black-box settings. They propose two approaches including (1) attack via flipping reward signals: attacker can flip the reward signal for a certain number of time steps, resulting that the victim agent is delayed from learning optimal actions; (2) attack via manipulating state information: the attacker can introduce false positive or false negative values into the next state, resulting in preventing the agent from taking the next optimal action. These proposed attacks

are evaluated in Software-Defined Networks, which are effective in luring the DRL agent to take sub-optimal actions.

Hussenot et al. 2020 [43] propose a targeted attack approach that is capable to fully control the DRL agent's behaviour. An attacker manipulates an agent's observation (that is, their perception of the environment) in order to consistently induce the agent to follow a target policy. When compared to previous work that directly changes the agent's state (i.e., the representation of the environment), the proposed attack is more realistic because white-access to the agent's inner working is not required. Moreover, the attack consists of a finite set of state-independent masks, which can be computed quickly and applied in real-time. In experiments, it is shown that the proposed approach is effective in attacking DQN in Atari games in white-box settings. It is also extended to black-box settings as well as the complex environment with realistic observations.

Sun et al. 2021 [18] propose a strategic poisoning algorithm against policy-based DRL, capable of achieving both non-targeted and targeted attacks. The authors propose a hybrid poison approach in which the attacker can poison either the agent's state, its actions or its rewards. This approach does not require any prior knowledge of the environment. In addition, the authors propose a new metric to measure and compare the vulnerability of RL algorithms under a variety of scenarios in order to characterize the stability of these algorithms. Their experiments involve state-of-the-art policy-based DRL algorithms (including A2C, PPO, and VPG) in a variety of environments. This attack has been shown to reduce a training agent's overall reward or force it to choose a specific policy within a limited budget or power, either in a white-box or black-box setting.

2.2.3.2 Action Space Poisoning

Yeow et al. 2019 [44] investigate adversarial attacks on the DRL agent's action space, aiming to minimize long-term discounted rewards by selecting bounded action space perturbations. The work proposes two white-box attack algorithms, namely Myopic Action Space (MAS) and Look-ahead Action Space (LAS). MAS distributes the attacks across the dimensions of the action space. At each time step, the attacker designs perturbations greedily without regard to the future. In contrast, LAS distributes attacks across both action space dimension and temporal

dimension, where the attacker looks ahead and constructs a sequence of perturbations with consideration of future rewards. Based on decoupling constraints MAS and coupled constraints LAS, the authors construct the attack algorithm as an optimization problem for minimizing the cumulative reward. In this study, the authors evaluate the proposed attack algorithms on PPO and DDQN, using continuous action environments from OpenAI's gym. Based on the results, LAS attacks are more effective than MAS with the same budget, as LAS uses the dynamics of the DRL agent explicitly. Further, the authors demonstrate that the LAS attack algorithm can be used to identify vulnerabilities in an agent's action space and design appropriate defenses.

2.2.3.3 Reward Poisoning

Ma et al. 2019 [14] investigate policy poisoning in batch RL agents, with the aim of forcing the agent to learn a target policy by slightly modifying rewards in the training dataset. The authors present an optimization framework for characterizing batch policy poisoning and provide a theoretical analysis of attack feasibility and cost. This work is developed under the assumptions that an attacker can access the original training data and that the attacker knows the RL agent's algorithms that are tabular certainty-equivalence and linear quadratic regulator. Based on the results of the experiments, the proposed attacks were successful in luring the agent to learn a target policy designed by the attacker, with small modifications to reward data.

Huang et al. 2019 [45] studies the malicious falsification of reward signals which can lead an RL agent into learning a target policy. This work focuses on investigating the adversarial behaviour of Q-learning agent, and establishes a theoretical framework to analyze the effect of attacking the Q-learning algorithm. The study examines the relationship between the Q-factors and falsified costs as well as the convergence of the Q-learning algorithm under various conditions. It then provides a robust region in terms of cost, in which the attacker is unable to achieve the desired policy. In addition, it characterizes conditions that can force the agent into the target policy as a result of falsified costs. The authors discuss four types of attackers according to their knowledge, including the omniscient attacker, the peer attacker, the ignorant attacker and the blind attacker. Experimental results show

that all these attackers can mislead the agent into learning the target policy via falsifying cost at each state.

Zhang et al. 2020 [15] investigate reward-poisoning attack on RL at training time, assuming that the attacker has complete knowledge of the environment model and the RL agent's algorithm. The authors focus on the RL agent that performs basic Q-learning and strive to force the agent to follow a target policy. They propose attacks which depend on the victim agent's Q tabular, and provides robust certificates even when the victim's underlying Q tabular does not converge. In addition, this work regards the attacker as an RL agent, thus, it formulates it as a high-level MDP and uses a deep RL algorithm (i.e., TD3) to obtain the high-level attacker's strategy. According to the experimental results, the combined FAA and TD3 perform much better than either the manually designed FAA or the pure TD3.

Rakhsha et al. 2021 [17] explore reward-poisoning attacks on RL in a black-box setting, aiming to enforce a target policy without knowing the environment or algorithm of the agent. An explore-and-exploit attack, U2, is proposed, which requires agents to follow a no-regret RL strategy. With the attack U2, RL agents can be hacked to efficiently gather information about the environment, allowing the attack U2 to carry out a targeted attack that is near-optimal. The results show that U2 can achieve an attack cost that is not significantly worse than the optimal white-box attack with the appropriate choice of hyperparameters.

2.2.3.4 Environment Poisoning

Bai et al. 2018 [46] interfere with the DQN agent in learning policy for automatic pathfinding, to degrade the training efficiency. In this work, the DQN agent first learns a policy that can solve the PathFinding problem. Then, the attacker analyzes the victim's learned policy and identify the weakness presented in the Q-value curves. Based on these weakness characteristics, the attacker constructs adversarial examples by adding obstacles to the environment. It has been demonstrated that this white-box attack method can hinder the robot's ability to learn the path through the maze and can result in a significant reduction in the agent's performance.

Chen et al. 2018 [47] also study attacks on A3C agent in automatic PathFinding tasks. They intend to prevent the agent from obtaining the path to the destination, or to severely delay the agent from finding the path. The authors proposed an attack method that can obtain dominant adversarial examples in any given maps. Essentially, this attack method involves adding "battle-like" obstacles to the gradient band where the value gradient arises most rapidly. The added obstacles change the A3C agent's environment and cause the agent to mess with its local information. Experiment results show the proposed attack method performs successfully at least 99.91% of the time, which demonstrates the method can generate valid adversarial examples with high confidence.

Rakhsha et al. 2020 [16] studies environment-poisoning attack which manipulates either rewards or transition dynamics, with the objective of enforcing an attacker-desired policy on the RL agent. This work presents attack approaches against an offline agent that plans behaviour based on dynamics programming (e.g., value iteration [25]), and against an online agent that takes actions based on regret-minimization framework (e.g. UCRL algorithms [48]). In this work, the authors establish an optimization framework to acquire an optimal attack with minimized attack cost. Furthermore, they provide theoretical analysis of conditions that ensure the success of an attacker and generate upper and lower bounds on the cost of an attack. Finally, these theoretical statements are supported by numerical simulations in this study. Simulation results indicate that the attacker was successful in forcing the agent to execute the target policy at a minimal cost.

2.2.4 Discussion

This section summarizes the aforementioned works and discusses how our study differs from these existing attacks.

Test-time Attacks and Training-time Attacks against RL. Different from test-time attacks which perturb the performance of a well-trained policy, training-time attacks aim to poison the learning of an RL policy, i.e., enforce an attacker-desired policy on the RL agent. Existing training-time attacks mostly poison the agent's policy by manipulating rewards or environment dynamics at training time [14–16]. For example, Ma et al. [14] poison rewards in the training set for the batch

RL agent, and Zhang et al. [15] study adaptive reward-poisoning attack on the Q-learning RL agent. Rakhsha et al. [16] poisons either reward values or transition functions to attack RL agents performing in cyclic tasks (i.e., the tasks without termination states). These existing training-time attacks succeed in forcing an RL agent to learn an attacker-desired policy, but they assume a significant access to the RL agent's perception (e.g., perceived rewards) or environment models (e.g., transition functions). Alternatively, we poison the training environment hyper-parameters (e.g., friction), which are external to the RL agent. Thus, the proposed environment poisoning attack can manipulate the RL agent's policy without any access to the RL agent's perception or the environment dynamics model. Moreover, unlike those training-time attacks [14–16] which are designed for a specific RL algorithm, our attack is proposed for a model-free RL agent but it is not restricted by the types of the RL algorithms.

In addition, it is important to mention that a clear division between training and testing is not always the case in reinforcement learning, especially for the lifelong RL (also known as continual RL) [49, 50]. In lifelong RL, the agent faces a series of consecutive tasks/environments during its lifetime. It should be able to learn each task quickly by building upon the experience gained in previous tasks. For such an agent, policy training and testing alternate, and the trade-off between exploration and exploitation changes over its lifetime. While policy poisoning on a lifelong RL agent is an interesting research topic, in the current thesis we are targeting an RL agent performing in a single task. Nevertheless, the attack against a single-task RL agent, which has a clear division between the training time and the testing time, can be regarded as a basis for developing policy poisoning on lifelong RL agents in the future work.

Offline and Online Attacks. For existing training-time attacks, they are developed in two manners, namely, offline attacks and online attacks. For offline attacks, the attacker has full knowledge of the training data set. It only makes one decision on rewards manipulation, and then provides the poisoned training set to the RL agent for planning [14, 16]. In contrast, online attacks mean that the attacker sequentially manipulates the feedback signal when interacting with the RL agent [15, 16]. It requires the attacker to access the victim's learning transitions (i.e. current state, chosen action, next state and corresponding reward), to make attack decisions on-the-fly. Such attacks are generally adaptive to the victim's learning

progress, however, they are constrained by the assumption of white-box settings. Different from offline and online attacks, we propose an intermediate attack framework where the environment is sequentially poisoned at regular intervals during the victim's learning process. The attacker makes an attack decision responding to the victim's policy features while does not need to grasp every transition of the victim. The proposed setting allows us to extend training-time attacks to black-box settings where learning algorithms or policy models are unknown.

White-box and Black-box Attacks. Test-time attacks against RL have been investigated in both white-box [9–11, 41] and black-box attack settings [13, 43]. Specifically, white-box attacks assume that the attacker has full knowledge of the victim's learning algorithm and policy model. The attacker perturbs the agent's perceived states during the deployment of the victim's policy in order to alter its selection of actions. Recently, more works [13, 43] have examined black-box attacks that do not allow the attacker access to RL policy model. These works are developed based on a key concept, namely, transferability of adversarial examples (i.e., crafted malicious inputs) [34]. Here, 'transferability' refers to the fact that adversarial examples, which may deceive one model, can also affect another. It has been demonstrated that transferability is effective for models with different architectures or training sets as long as these models are trained for the same problem. Based on the transferability of adversarial examples, some approaches [13, 43] learn attacks on a proxy model and built black-box adversarial examples to fool deep RL agents.

For training-time attacks, researchers recently make a breakthrough in the development of realistic reward-poisoning attacks that can manipulate the RL policy *without* prior knowledge of the RL system [17, 18]. In particular, Rakhsha et al. [17] theoretically study a black-box reward-poisoning attack against no-regret RL algorithms. In turn, Sun et al. [18] propose a practical reward-poisoning algorithm for policy-based deep RL methods without knowledge of the environment. In this thesis, we pursue the same motivation as [17, 18] but focus on the development of realistic environment-poisoning attacks. We investigate a novel environment-poisoning attack that requires minimal attacker's prior knowledge of an RL system.

Environment Poisoning and Shaping. It has been shown in previous works that RL agents are significantly sensitive to the training environment [51], thus manipulating environment is an effective way to influence the agent's policy learning [52–55]. Generally, these works can be grouped as environment shaping and environment poisoning based on their objectives. Specifically, the environment shaping aims to help the agent speed up acquiring its optimal policy, which is popular in behaviour teaching area [52, 54]. The environment poisoning, on the contrary, enforces an arbitrary target behaviour on the agent. The target policy can not be the optimal one for the agent. It is usually studied in adversarial attack field [16]. Even though objectives are different, teaching and attacking are equivalent mathematically. This means our work can be used in teaching an agent desired behaviours.

When our work is discussed in the teaching area, it is important to note the difference between our work with the inverse RL [56, 57] due to their similar objectives. IRL performs teaching by human inputs, and requires the agent to be able to estimate the teacher's intention; while our teaching depends on automated environment design without the requirement of the learner's motivation. It is also necessary to note the difference between our work with curriculum learning (CL) [58, 59] due to their similar approaches. Some works in CL design a sequence of training environments without consideration of effort of environment changes; while our teaching considers the environment-modification price when optimizing the environment design. Such consideration of environment-modification effort is one main difference between our framework design with most existing teaching methods [59], but is one similar point between our work with works about environment shaping [52].

Environment Hyper-parameters. Now, it must be mentioned that there are RL approaches capable of identifying environment changes induced by hyper-parameter shifts, and constructing a robust behaviour strategy in the context of such changes [21–23]. The goal of these methods is to generate a behaviour useful across tasks and environments. However, they mostly disregard the possibility of a *constructive, strategic* adversary that modulates environment hyper-parameters. Thus, while we view robust algorithms as a key component in building white-box proxies in our future research, in the current thesis we make the standard relaxation assumption of training-time RL attacks. Namely, we assume that the attacked RL

agent is oblivious to the attack and continues to operate normally throughout the sequence of environment modifications.

2.3 Defenses against Adversarial Attacks on RL

This section provides a brief overview of defenses against test-time attacks on RL, followed by a detailed description of defenses against training-time attacks.

2.3.1 Defenses against Test-Time Attacks

In this part, we briefly review defense approaches against test-time attacks on RL, which enables an RL policy to be empirically robust against perturbed state inputs during test time. Referring to the survey [60], there are several categories of these defenses as illustrated in Figure 2.6.

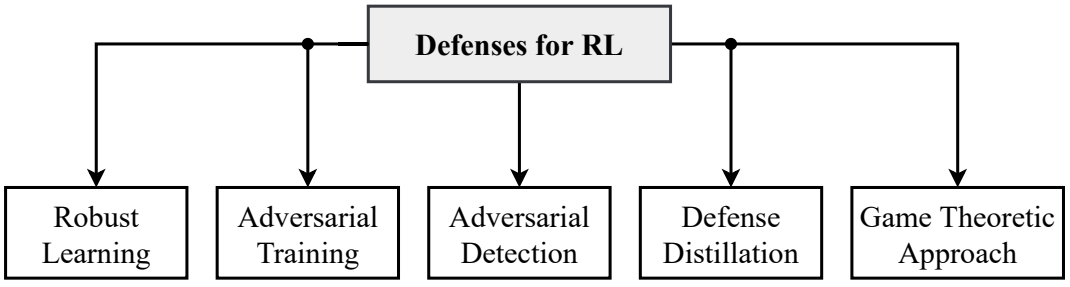


FIGURE 2.6: Categories of defenses against test-time attacks on RL.

- **Robust Learning** [61][62][63]. The concept of robust learning refers to a training mechanism that ensures robustness against adversarial attacks during the training process. The learned policies can maximize the cumulative rewards even when flawed, or even adversarial, observations are present during the test time.

- **Adversarial Training** [11, 35, 64]. An effective technique for improving policy robustness is adversarial training, which involves augmenting training data with adversarial examples to retrain the policy. In this way, the policy can learn a better distribution, resulting in an improvement in generalization beyond the training manifold.

- **Adversarial Defense** [65–67]. Adversarial defense can be described as two steps: 1) a model is specifically trained to detect adversarial attacks, and 2) the robustness of the policy is ensured by segregating the true training samples from the adversarial ones. Thus, the policy decision can not be influenced by adversarial inputs since they are ignored.

- **Defense Distillation** [68, 69]. defense distillation is a method of extracting the policy from a dense network that can be applied to the training of another, smaller network that can make expert decisions. In this way, the number of attackable parameters is reduced, thereby providing greater stability and resistance to adversarial noise and attacks. In order to achieve optimal results, this solution should be combined with other techniques, such as adversarial training and adversarial detection.

- **Game Theoretical Approach** [70] [71] [72]. It is possible to formulate the challenge of robust policy learning as a zero-sum minmax objective function, which is intended to ensure robustness to variation in testing and training conditions, even when an adversary is present.

2.3.2 Defenses against Training-Time Attacks

The purpose of this section is to review recent developments in RL defenses against training-time attacks. A summary of these defense methods can be found in Table 2.3

TABLE 2.3: Summary of defences against training-time attacks on RL.

Paper	**Targeted Training-Time Attacks**				**Targeted Settings**
	observation	**reward**	**transition**	**trajectory**	
Behzadan el at. [73]	✓				DRL
Behzadan el at. [74]	✓				DRL
Banihashem et al. [75]		✓			Ergodic MDP
Wang et al. [76]		✓			DRL
Lykouris et al. [77]		✓	✓		Tabular MDP, Linear MDP
Chen et al. [78]		✓	✓		Tabular MDP
Wei et al. [79]		✓	✓		Finite-horizon linear MDP
Zhang et al. [80]		✓	✓		Tabular MDP, DRL
Zhang et al. [81]				✓	Offline RL
Wu et al. [82]				✓	Offline RL

2.3.2.1 Defenses against Observation Poisoning

Behzadan et al. [73] investigate the robustness and resilience of deep RL to perturbed observations during training time and test time. In their study of training-time attacks, the perturbation probability for each observation must be below a certain threshold for DQN to recover from deteriorating training performance. Thus, the authors suggest that a critical limit needs to be reached in terms of the number of adversarial samples stored in a memory in order for an agent to recover from adversarial attacks. This enables the agent to learn the perturbation statistics from a random batch sampled from memory. They demonstrate empirically that DQN can recover from noncontiguous training-time attacks by adjusting its policy in response to adversarial perturbations. The results indicate that policies, which are trained under adversarial perturbations, are more robust against test-time attacks.

Behzadan et al. [74] examines a method for mitigating policy manipulation attacks via intentionally perturbed inputs (i.e., adversarial examples), at both test time and training time. A parameter-space noise exploration is proposed as a means of mitigating such attacks on the DQN network. Specifically, the researchers compare the resilience of two DQNs to adversarial attacks: 1) a DQN based on ϵ-greedy policy learning and 2) a DQN based on a parameter-space noise exploration technique (i.e., NoisyNets). They demonstrate empirically that NoisyNets make DQN more resilient to training-time attacks. A noisy DQN architecture is also more resistant to black-box attacks at training time and less susceptible to transferability of adversarial examples at test time, compared to a plain DQN architecture.

2.3.2.2 Defenses against Reward Poisoning

Banihashem et al. [75] investigate robust RL against reward-poisoning attacks [14–16] that seek to force learning an attacker-desired policy while minimizing the cost of reward manipulation. The authors formulate the defense task as an optimization problem for optimizing the worst case performance of the agent, among a range of plausible candidates for the true reward function. In this study, two specific settings have been taken into account, namely the agent's knowledge of the attack parameter and lower bounds on the effectiveness of the defense policy.

Wang et al. [76] investigate a robust RL framework for dealing with biased noisy rewards during training. The authors propose an unbiased estimator of true rewards under RL conditions. With the proposed estimator, an RL agent can learn in noisy environments in which only perturbed rewards are observed. Specifically, the core idea of their solution consists of two parts: 1) the estimation of a reward confusion function that generates rewards for helping the RL agent to learn when the rewards are perturbed; 2) the definition of unbiased surrogate rewards based on those perturbed rewards. Using Q-learning as an example, they empirically demonstrate the convergence to optimal policy and the complexity of samples. Furthermore, they validate the proposed technique on DRL platforms (i.e., OpenAI Gym). Results show that the surrogate reward rescue policies from misleading rewards and achieve higher expected rewards with faster convergence.

2.3.2.3 Defenses against Both Reward and Transition Poisoning

Lykouris et al. [77] presents the first study of robust RL under both adversarial rewards and adversarial rewards during training time. For both linear and tabular MDPs, they design an algorithm that achieves a instance-dependent regret bound for an adversarial setting in which a certain number of episodes are corrupted. Additionally, Chen et al. [78] provide more strict bounds for total corruptions for tabular MDPs, while Wei et al. [79] achieve worst-case optimal bounds without knowing the total amount of corruption for linear MDPs over a finite horizon.

Zhang et al. [80] investigate the issue of robust RL in the face of an adaptive adversary that is capable of arbitrarily poisoning both rewards and transitions in ϵ fractions of all learning episodes. They develop a variant of natural policy gradient approaches that is adaptive to the contamination level without knowledge of the fraction ϵ. They theoretically present a guarantee to seek $O(\epsilon^{1/4})$-optimal policy in the presence of strong data corruption. They furthermore implement their approaches in neural networks and empirically showing its strong robustness performance in high-dimensional RL problems, such as the Mujoco continuous control tasks.

2.3.2.4 Defenses against Data Corruptions

Zhang et al. [81] investigate offline-RL's robustness when data corruption occurs. The authors examine the situation where an adversary can modify the ϵ fraction of a batch dataset composed of tuples (s, a, s', r), with the objective of enabling an agent to learn a policy that is near optimal. Through theoretical analysis, they propose robust variants of the Least Square Value Iteration algorithms as well as provide general robustness bounds for RL. The analyses are based on the assumption of a linear MDP or a bounded distance between the learned and Bellman optimal policies.

Wu et al. [82] study the robustness of off-line reinforcement learning against a threat model of general poisoning attacks, where the attacker is permitted to manipulate training trajectory arbitrarily. They propose a certification framework and two certification criteria that include per-state action stability and cumulative reward bound. The framework is used to determine how many poisoning trajectories can be tolerated in light of the limited knowledge about the attack based on different certification criteria. In addition, they present a novel partitioning and aggregation protocol for the training of robust policies.

2.3.3 Discussion

In light of the literature reviews discussed above, the majority of existing defenses against training-time attacks are designed from a robustness perspective. There are two general categories of these works: (1) Some works [61, 75, 77–80, 82] offer theoretical guarantees concerning the policy learning under perturbations within a certain range; (2) Other studies [73, 74, 76, 80] empirically examine the robustness of the training model to perturbations. Most of these works are investigating an agent's ability to maintain its desired functionality despite perturbations on rewards or transitions. However, very few studies focus on the defenses against training-time attacks from a resilience perspective, namely, recovering a policy from malicious manipulations. One particular interest is the work proposed by Behzadan et al. [73, 74], which investigate the resilience of deep RL to perturbed observations during training time. Unlike these existing works, this thesis investigates a policy-resilience solution against training-time environment poisoning, which an be considered as a complement to imperfect robust solutions.

Additionally, of particularly interesting topic is safe RL [83]. The safe RL attempts to ensure the reasonable performance of an RL system, and/or respect safety constraints during the learning and deployment of RL policy. Even though both the safe RL and the robust/resilient RL aim to protect the RL system from *risks*, their targeted risks are different. Specifically, the safe RL focuses on addressing the risk which refers to the dangerous issues in the physical system (e.g., the cliff for autonomous cars) or the inherent uncertainty in the environment (i.e., with its stochastic nature). It aims to reduce the agent damage or injury when learning the policy, generally by transforming the optimization criterion or modifying the exploration mechanism. In contrast, the robust/resilient RL aims to ensure an RL agent to learn and deploy an optimal policy in the event of adversary perturbations.

Chapter 3

Transferable Environment Poisoning Attack

3.1 Introduction

Proliferation of applied RL techniques has drawn the ever increasing attention to their security issues. In order to examine an RL agent's susceptibility to security threats, researchers are developing a research substream that aims to subvert its policy during training [14–16, 41]. Admittedly, these methods are successful to a degree, and propel the RL agent towards a target policy which is designed by the attacker (i.e., attacker-desired policy). However, several strong assumptions wrt access to the RL agent's perceptions and knowledge of the agent's policy greatly limit the applicability of these approaches.

In more detail, existing training-time attack approaches [14–16] assume that the attacker has full knowledge of the RL agent's learning algorithm and policy model. Furthermore, it is assumed that the agent's perceptions and memory are accessible, so that the complete interaction information (e.g., rewards) between an agent and its environment can be manipulable. Thus, reward poisoning is a common weapon of choice to interfere with the agent's learning process and drive it towards the attacker-desired policy. However, the agent's perceived rewards are determined by both external feedback and the agent's *intrinsic* environment [25]. For example, in a classical story 'Phil prepares his breakfast' [25, 84], Phil's chosen action might receive different rewards depending on his hunger level, his mood and other features

of his body, which are internal and private properties of the agent. Similarly, some robots' perceived rewards are determined by the configuration of embedded sensors, encapsulated away from attackers. In such scenarios, reward poisoning [15, 85, 86] is infeasible, as there is no access to the manner in which rewards are generated. Similar failure easily befalls approaches that work by poisoning the RL agent's memory [14]. Thus, it is necessary to design an attack approach that works only through properties of the environment (i.e., environment hyper-parameters) that are external to the RL agent and determine how the environment response to agent's actions (i.e., environment dynamics). Additionally, even though we may have some design documentation on the agent's learning algorithm, it is important to design an attack approach that will be as independent from such information as possible.

Specifically, we develop an approach to automatically design a transferable attack strategy, trained in a white-box setting with a proxy agent and then applied to a white-box or black-box victim agent. In more detail, we deploy a bi-level Markov Decision Process (MDP) framework as a means of finding a training-time attack strategy by poisoning environment dynamics. At a lower level, an RL proxy agent explores an MDP environment, seeking to maximize its expected cumulative rewards. At a higher level, the attacker seeks to tweak the dynamics of the lower-level MDP to drive the proxy agent towards an attacker-desired policy. We assume that the tweak can be generated at regular intervals with respect to the lower-level timeline, and that prior to the tweak the higher-level attacker obtains a reliable estimate of the proxy agent's current policy. While the attacker's main goal is to drive the proxy agent towards acquiring the attacker-desired policy, the attacker is also limited in its ability to change the proxy agent's environment. In fact, we relate the degree to which the environment has been changed to the attacker's *effort*. Thus, we build the attack optimization around the combined deviation of (a) the environment dynamics from their natural form *and* (b) the proxy agent's policy from the attacker-desired policy. The deviation measurement is based on the Kullback-Leibler Divergence Rate [87] between the actual RL system (i.e., consisting of the poisoned environment and the proxy agent's policy) and the attacker-desired system (i.e., consisting of the natural environment and the attacker-desired policy).

We demonstrate the effectiveness of the proposed approach in two stages. First,

we develop it to handle white-box scenarios, where the victim agent's algorithm is known to the attacker; hence, the proxy agent is precise in simulating the victim of the attack. We then show, how the approach can be extended to black-box scenarios, where the proxy agent approximates the victim using the latter's observed interaction trajectories. Additionally, we explore the transferability of the environment-poisoning attack strategy between RL agents by explicitly creating scenarios where the proxy agent and the victim run different algorithms. Furthermore, using the transferability property, we design a population-based attack cost to seek an efficient and synchronous policy manipulation over a population of independent and various RL agents.

Overall, the contribution of this chapter can be summarized as follows [1]:

- We propose a transferable environment-dynamics poisoning attack (TEPA) against an RL agent at training time, seeking to force an RL agent to an attacker-desired policy via minimized environment changes.
- We present a close-loop attack design framework based on a bi-level Markov Decision Process architecture, showing that the architecture can be successfully resolved using a DRL technique that generates valid attack strategies.
- We evaluate the effectiveness and efficiency of TEPA against tabular-RL agents, including Q-learning, Sarsa and Monte Carlo agents, in navigation tasks with discrete state domains.
- Our empirical study investigates the transferability of the environment-poisoning attack, demonstrating that our approach can be applied to black-box RL agents, as well as a population of RL agents.

3.2 Preliminaries

The development of this work is inspired by Behaviour Cultivation (BC) [53]. In this section, we describe the core idea of BC and how it fits into our proposed attack approach.

[1] The work in this chapter has been published in Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'21).

Behaviour Cultivation. The goal of BC [53] is to induce an RL agent to learn a desired behaviour with minimized changes of environment dynamics. In BC, the learner has no intention of following the desired behaviour, instead, it seeks to maximize its own benefit. The teacher aims to induce the learner to acquire an arbitrary behaviour that is desired by the teacher. These characteristics allow BC to be effective in adversarial attacks on an RL agent's policy. In this work, we borrow the core idea of enforcing an arbitrary desired policy via environment-dynamics tweaks from BC. We extend this core idea into a closed-loop control sequence of environment-dynamics modifications responding to the learner's actual learning progress.

Since BC aims to achieve a desired behaviour on the learner via minimized environment-dynamics changes, it considers the cost of changing dynamics when minimizing the deviation between the learner's actual policy with the desired one. BC uses Kullback Leibler Divergence Rate (KLR) [87] as the measurement of this deviation. In this work, we also adopt KLR to measure the attack cost to achieve attack success with control of the attack effort.

Kull-Leibler Divergence Rate. Kullback-Leibler divergence measures how one probability distribution is different from a reference probability distribution.

Definition.1: Given discrete probability distributions p and q, the Kullback-Leibler divergence of q from p is

$$D^{KL}(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

When Kullback-Leibler divergence is extended to Markovian processes, it is called Kullback-Leibler Divergence Rate [87].

Definition.2: Given Markov Processes X_t^1 and X_t^2, the Kullback-Leibler Divergence Rate (KLR) is

$$D_{KLR}(X^1||X^2) = \lim_{n\to\infty} \frac{1}{n} D^{KL}(P(X^1 = x_n)||P(X^2 = x_n))$$

When the Markov Process is described by two conditional transition matrices, $P(x'|x)$ and $Q(x'|x)$, the KLR is proven to be

$$D_{KLR}(P||Q) = \sum_{x} D^{KL}(P(x'|x)||Q(x'|x)) \times p(x)$$

where $p(x)$ is the stationary distribution of P [87].

3.3 Methodology

In this section, we first present the design of attack framework and then introduce the attack optimality criteria in white-box settings, followed the description of attack strategy learning.

3.3.1 Attack Framework

In this part, we formulate our environment-poisoning attack problem using a bi-level MDP architecture, which is shown in Figure 3.1. Here the victim is an RL agent who interacts with the environment, seeking maximized cumulative rewards. The attacker is also an RL agent who regards the victim as a dynamic system. At intervals of the victim's learning process, the attacker manipulates the victim's environment dynamics, responding to the victim's momentary policy information and environment conditions, as shown in Figure 3.2. The attacker's objective is to learn a strategy which succeeds in enforcing an attacker-desired policy on the victim agent via minimized changes in environment. In the following, we will introduce the victim-level MDP and attacker-level MDP in more detail.

Victim-level MDP. The victim's Markovian environment is represented by the tuple $\langle S, A, T_{\hat{u}}, r, q_0, \gamma \rangle$. Here, $s \in S$ is an environment state and $q_0(s)$ is a distribution over initial states. A is the set of the victim's available actions and $r : S \times A \times S \to \mathbb{R}$ is the reward function. $\gamma \in (0, 1)$ is the discount factor. $T_{\hat{u}} : S \times A \times S \to [0, 1]$ is the probabilistic state transition function with tweaks of hyper-parameters coming from space $\hat{U}$ (described in the attack-level MDP). Thus, $T_{\hat{u}}(s'|s, a)$ represents the probability of the environment state changing from s to

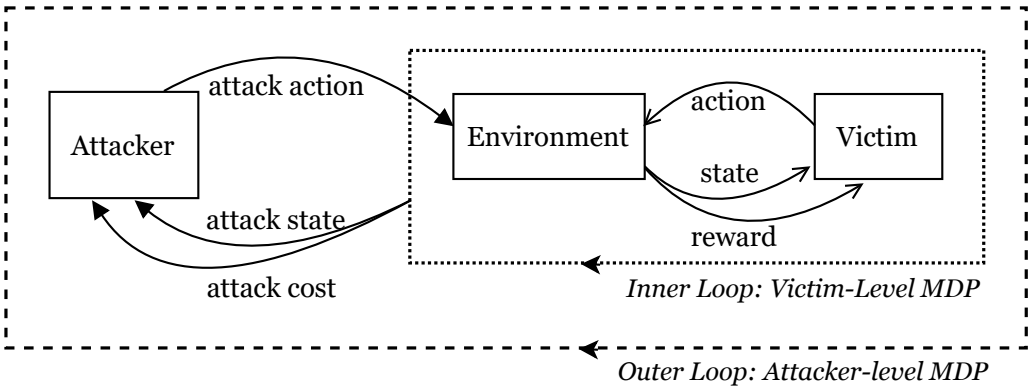


FIGURE 3.1: Attack framework: bi-level Markov Decision Process.

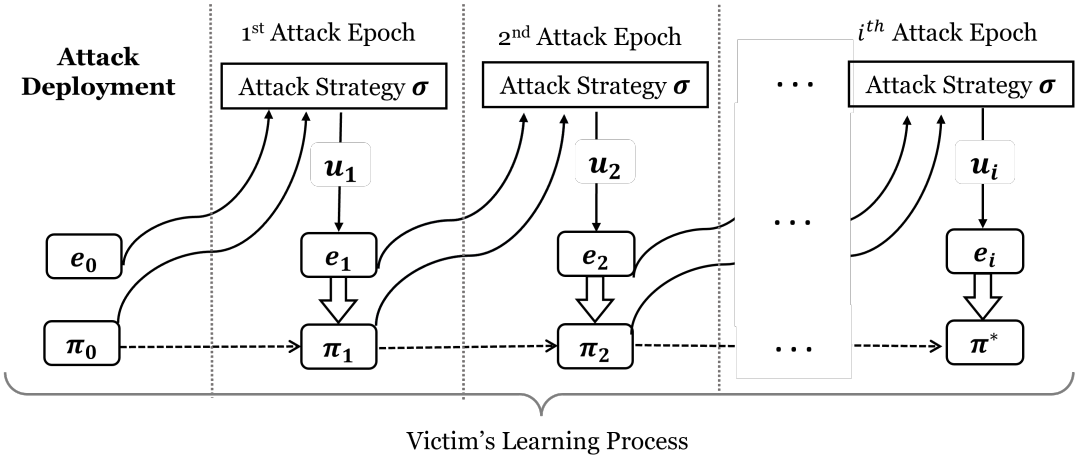


FIGURE 3.2: Illustration of environment-poisoning attack.

s', given that the environment dynamics was tweaked by $\hat{u} \in \hat{U}$ and the victim chose the action a. The victim's objective is to find an optimal policy $\pi(a|s)$ which maximizes the cumulative discounted rewards.

Attack-level MDP. We define the attacker's environment as a higher-level Markovian process, which is denoted as the tuple $\langle \Theta, U, F, c, \theta^*, \gamma_a \rangle$ where:

- Θ is the attacker's state space. $\theta \in \Theta$ represents the victim agent's policy and θ^* denotes the target policy designed by the attacker. In this chapter, we focus on the attack against a tabular-RL agent. Thus, the RL policy is represented as a tabular table (e.g., Q table). Namely, θ can be a Q table, accordingly, Θ is a continuous set.
- U is the attacker's action space. $u_i \in U$ is the tweak to the victim's environment hyper-parameters. Here, the hyper-parameters control how the

environment responds to the victim's action. u_0 means that environment hyper-parameters are not changed by the attacker. $u_{1:i}$ represents the aggregate outcome of a sequence of tweaks, which is also termed as $\hat{u}_i$ within confines of a set $\hat{U}$.

- F is the attacker's environment transition probability function. It captures how the victim updates its policy. $F(\theta'|\theta, \hat{u})$ represents the probability that the victim changes its policy parameter from θ to θ' in the environment tweaked by $\hat{u}$.
- $c : \Theta \times \hat{U} \to \mathbb{R}$ is a function representing the attack cost. It denotes the combined impact of the victim's (possibly undesirably) parameterised policies and the attacker's aggregate tweaks in the victim's environment.
- γ_a is the discount factor.

The attack objective is to find an attack strategy $\sigma(u_i|\theta_{i-1}, u_{1:i-1})$ which can force the victim to acquire the attacker-desired policy while minimizing cumulative changes to the environment dynamics. The attacker aims to minimize the cumulative discounted attack cost

$$C = \sum_{i=1}^{\infty} \gamma_a^i c_i, \tag{3.1}$$

where i is the attack epoch and the attack cost c_i will be defined mathematically in the following.

3.3.2 Attack Learning

In this part, we present our design of the attack optimality criteria, and describe the training process of an attack strategy in white-box settings. Note that the white-box setting is classic and common in the research area of adversarial attack [9, 14–16]. When we talk about white-box attacks, the attacker is assumed to know the agent's learning algorithms (i.e., learning algorithm and policy model) and its environment dynamics model.

Before describing the attack optimality criteria, we first introduce the target policy which designed by the attacker in a discrete state domain. Given a target state set $S^* \subseteq S$, the target policy is denoted as

$$\begin{cases} \pi^*(a^*|s) = 1 & \text{s} \in S^* \\ \pi^*(a|s) = \pi_i(a|s) & \text{s} \notin S^*, \end{cases} \tag{3.2}$$

where a^* is the target action desired by the attacker and $\pi_i(s)$ is the agent's actual policy. This is a partial target policy where target actions are defined for S^*. Compared with the complete target policy (i.e., desired actions are defined for all the states) proposed in BC [53], the partial target policy is more practical and applicable for large-size discrete domains.

Based on the definition of the target policy π^*, we design the attack cost c_i at each epoch i, as described by the following:

$$\begin{aligned} c_i(\theta, u) &= D_{KLR}(P_i||P^*) \\ &s.t. \\ D_{KLR}(P_i||P^*) &= \sum_{s,a} q_i(s)\pi_i(a|s;\theta)D_i^{KL}(s,a) \\ D_i^{KL}(s,a) &= \sum_{s',a'} P_i(s',a'|s,a)\log\frac{P_i(s',a'|s,a)}{P^*(s',a'|s,a)} \\ P_i(s',a'|s,a) &= T_{u_i}(s'|s,a)\pi_i(a'|s';\theta') \\ P^*(s',a'|s,a) &= T_{u_0}(s'|s,a)\pi^*(a'|s';\theta') \\ q_i(s') &= \sum_{s,a} q_i(s)\pi_i(a|s,\theta)T_{u_i}(s'|s,a). \end{aligned} \tag{3.3}$$

Here, $q_i(s)$ is the stationary distribution of the agent's environment MDP, which is known by the attacker. $P_i(s',a'|s,a)$ represents a stochastic process over state-action pairs, where an agent follows the policy $\pi_{\theta'}(a|s)$ in the environment changed by a sequence of tweaks $u_{1:i}$, where $u_j \in U$ for all $j \in [1:i]$. Similarly, $P^*(s',a'|s,a)$ is the ideal Markovian Process from perspective of the attacker.

Note that Equation 3.3 defines attack cost computation at each attacking epoch i. The attacker and the victim have different time scales. At each attacking epoch i, the environment dynamics parameterization u_i is fixed. The victim interacts

with the poisoned environment within T_{max} time steps and learns policy π_i. Then, the attacker recognizes the parameterization of the learner's updated policy and accordingly manipulates environment hyper-parameters, starting another attacking epoch.

With the definition of attack cost (i.e., Equation 3.3), we can learn an attack strategy to minimize the cumulative discounted cost as Equation 3.1, using deep-RL algorithms. In this work, we choose Deep Q-learning (DQN) [88] in discrete attack action domain, and adopt Twin Delayed DDPG (TD3) [31] for continuous attack actions.

Algorithm 4 Training of Attack Strategy in White-box Settings

1: Given
2: dynamics model of natural environment T_0
3: victim's initial policy π_0
4: attacker-desired policy π^*
5: compute ideal Markovian process $P^* \leftarrow \pi^* \times T_0(u_0)$
6: **for** episode_num = 1, 2, 3, ... I_{max} **do**
7: reset environment T_0 and victim's policy π_0
8: obtain attacker's initial state $x_0 \leftarrow \{\pi_0, T_0\}$
9: **for** attack epoch i=1, 2, 3, ... **do**
10: $u_i \leftarrow \sigma(x_{i-1})$ ▷ choose attack action
11: $T_i \leftarrow T_{i-1}(u_i)$: ▷ poison environment dynamics
12: $\pi_i \leftarrow f_{victim}(T_i, \pi_{i-1})$: ▷ victim learns policy
13: $x_i \leftarrow \{\pi_i, T_i\}$: ▷ attacker's input is updated
14: $P_i \leftarrow \pi_i \times T_i(u_i)$: ▷ compute actual Markovian process
15: $c_i \leftarrow D_{KLR}\ (P_i, P^*)$: ▷ compute attack cost
16: $B \leftarrow \{x_{i-1}, u_i, c_i, x_i\}$: ▷ save transitions to replay buffer
17: update attack strategy network σ
18: **if** $\{\max_a \pi_i(a|s) == \pi^*(a^*|s) | s \in S^*\}$ **then**
19: attack is done, go to the next episode
20: **end if**
21: **end for**
22: **end for**

The learning of an attack strategy is summarized in Algorithm 4. Here, line 10 $\sim$ 11 introduce how the attacker poisons the environment dynamics, and line 12 means that the victim agent learns policy in the poisoned environment using algorithm f_{victim}. In line 14 $\sim$ 15, the attack cost is computed following Equation 3.3. Then, the attacker's transitions are saved in replay buffer B for optimizing the strategy network. Finally, the attack strategy σ is updated using samples in the replay buffer. When the victim chooses desired actions in all the target states or the

number of attacking epoch reaches I_{max}, the attack episode is terminated and training goes to the next episode.

3.4 Attacks on a Black-Box RL Agent

In this section, we investigate how to apply TEPA on a black-box RL agent in two scenarios. First, the attacker has no prior information about the RL agent's learning algorithm, i.e., how the agent learns the policy, while it is assumed to know the victim agent's policy model, i.e., what the agent's momentary policy is. In the second scenario, the attacker knows nothing about the victim agent's learning algorithm as well as its policy, instead, the attacker can only observe the agent's behaviour trajectories. Compared to those white-box attacks [14–16], these two scenarios are closer to real-world situations.

Transferable Attack. In the first scenario, the challenge is how the attacker designs its attack strategy without information about the victim's learning algorithm. To address this issue, we would like to demonstrate the *transferability* of the poisoned environment first. As introduced in [25], comparing with other types of machine learning, the most significant characteristic of RL is that it uses feedback from the training environment to evaluate the action taken, rather than instructed by given correct actions. Thus, the interaction with the training environment plays a crucial role in determining the RL agent's optimal policy. As for the agent's learning algorithms or policy representations, they only affect how efficiently the agent finds the optimal policy, instead of determining what the optimal policy is. Therefore, the training environment which poisons one agent's policy should be transferred to attack other RL agents, even if these agents utilize different learning algorithms or policy representations. We term it as the transferability in poisoned environments.

We utilize the transferability in poisoned environments to solve the challenge in the first black-box setting. Here, we propose that the attacker trains its attack strategy on a white-box proxy agent (i.e., the learning algorithm is known by the attacker), and then transfers the strategy to attack black-box victim agents.

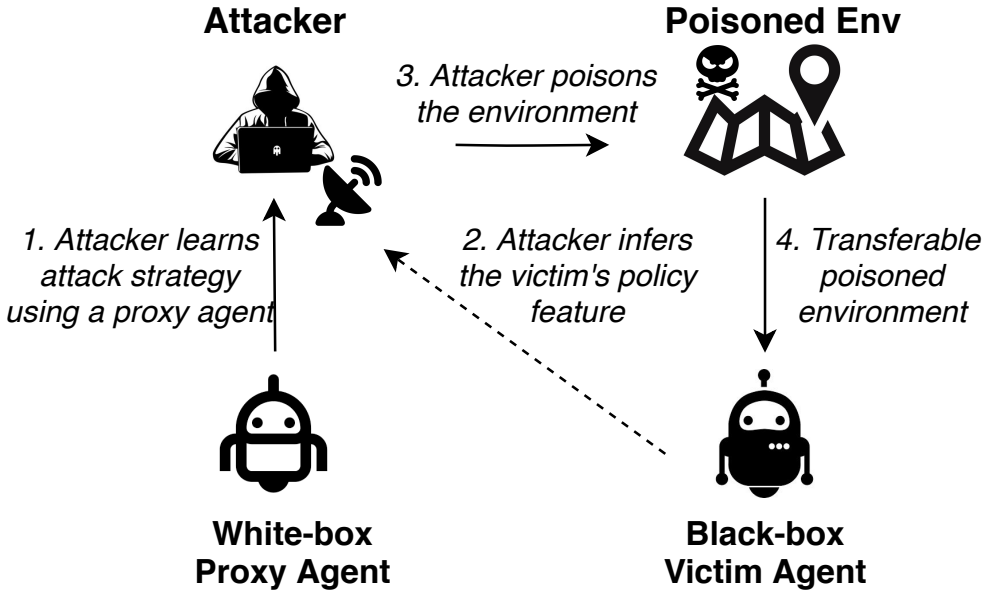


FIGURE 3.3: Illustration of the transferable environment-poisoning attack against a black-box RL agent.

Transferable Attack with Agent Modeling. Since the attacker is difficult to have prior knowledge of the victim's private information in real-world scenarios, the second scenario (i.e., both victim's learning algorithm and policy model are black-box) is more realistic and worth being investigated. Depending on the solution of the first scenario, here, the attacker's challenge is how to infer the victim's policy when deploying responsive attacks. Specifically, as described in Figure 3.3, the attacker learns the attack strategy on a white-box proxy RL agent and applies the transferable strategy to a black-box victim agent responding to its behaviour features. Here, the key task is how to obtain the victim agent' behaviour features according to its trajectories.

Inspired by generative policy representations [89, 90], we use the encoder-decoder neural network to obtain policy parameterisations which can best represent the victim's behaviour features. Specifically, at each attack epoch i, the attacker first collects the victim's successive N state-action transitions $\mathbf{X^i} = \{s_t^i, a_t^i\}$ derived from the victim's policy π_i, where $t \in [t_k, t_{k+N}]$. Then, the attacker learns a representation function $f_\omega : \mathbf{X^i} \to \mathbb{R}^d$ which encodes trajectories $\mathbf{X^i}$ as a d-dimensional real-valued vector embedding. This embedding vector represents the victim's policy feature. The decoder network is a policy function $f_\phi : S \times A \times \mathbb{R}^d \to [0, 1]$ which maps the victim's state and the vector embedding to the distribution over

its actions. The parameter ω and ϕ in the encoder-decoder network are learnt via maximizing the negative cross-entropy objectives:

$$\mathbb{E}_{\mathbf{x_1^i}\sim\mathbf{X^i},\mathbf{x_2^i}\sim\mathbf{X^i}}\left[\sum_{\langle s^i,a^i\rangle\sim\mathbf{x_2^i}}\log f_\phi(a^i|s^i,f_\omega(\mathbf{x_1^i}))\right] \tag{3.4}$$

Here, $\mathbf{x_1^i}$ and $\mathbf{x_2^i}$ are two different trajectories from the transition set $\mathbf{X^i}$, which are used to train encoder and decoder network respectively. Since the output of the encoder network (i.e. vector embedding) represents the victim's policy features, it is used as the input to the attacker.

In summary, our proposed TEPA can effectively poison an RL agent's policy without prior knowledge about its learning algorithm or/and policy model. This is achieved by the transferability property. Specifically, the attack strategy is learned on a white-box proxy agent, which can be zero-shot transferred to poison a black-box victim agent's policy. Importantly, the proxy agent and the victim agent should perform the same task in the same training environment, regardless of the type of their learning algorithms.

3.5 Attacks on a Population of RL Agents

As described in Section 3.4, our attack strategy, which is learned on a proxy RL agent, can be transferred to poison different victim agents in the same environment. Due to transferability, our attack strategy is additionally able to poison multiple RL agents that independently learn the same task in isolated environments. Namely, our environment-poisoning attack can further target a population of *independent* and *isolated* agents that may adopt different learning algorithms or policy models. The attack objective is to *synchronously* manipulate behaviours over the population via minimal cumulative changes in all the environments.

In this section, we first briefly describe the scenarios in which the population-based attack will be applied, followed by an introduction of the learning and conducting synchronous attacks on a population.

There exists works ([91] [92] [93]) studying security threats to multiple distributed agents in cyber-physical systems, such as autonomous vehicles, unmanned aerial vehicles and electrical grids. They focus on corruption of sensory data, manipulation of actuators' inputs or interruption of communication networks during deployment. However, the majority of existing training-time attacks focus on a single agent ([17, 18]), or even only threaten one specific RL algorithm ([14, 15]). Few works study training-time attacks on distributed / multiple RL agents in these systems. Of particular interest is that [16] design a reward-poisoning attack approach that targets a sequence of RL agents which take turns to learn their policies in one environment. In contrast, we study an environment-poisoning attack against a population of isolated RL agents that concurrently learn their policies in their individual environments. We aim to force all these agents to learn the same attacker-desired policy via minimized overall changes in environments. Imagine, for example, that distributed autonomous vehicles are learning how to handle heavy traffic. Our attack objective is to synchronously manipulate the learning of their driving policies by maliciously altering each individual vehicle's training environment.

This population-based attack cost at the attack epoch is designed to achieve synchronous poisoning on policies of a population through minimizing changes to the overall environments. We design the attack cost C_i at the i^{th} attack epoch as

$$\begin{aligned} C_i &= \sum_{n=1}^{N} c_i^n - \beta H(\omega) \\ &s.t. \\ c_i^n &= D_{KLR}(P_i^n, P^*) \\ H(\omega_i) &= -\sum_{n=1}^{N} q(w_i^n) \log q(w_i^n). \end{aligned} \tag{3.5}$$

Here, the population-based attack cost C_i consists of two terms, which are designed to achieve two aims: 1) to minimize overall deviations of all the agent's policies from the desired ones via minimizing overall attack efforts and 2) to accomplish synchronous poisoning over the population, i.e., ensure that all the agents' policies are manipulated in similar speeds across attack epochs.

Specifically, in the first term, c_i^n represents the attack cost of the n^{th} RL agent at the i^{th} attack epoch, which is defined as Equation 3.3. $\sum_{n=1}^{N} c_i^n$ measures the overall

attack performance of the population by accumulating c_i^n over all the agents. The second term $H(\omega_i)$ is the entropy of a distribution of attack performances within the population at the i^{th} attack epoch. Here, the distribution $q(\omega)$ is defined as

$$q(\omega_i^n) = \frac{e^{\eta \times \omega_i^n}}{\sum_j e^{\eta \times \omega_i^j}} \quad s.t. \quad \omega_i^n = \Delta(\pi_i^n || \pi^*), \tag{3.6}$$

which is determined by each agent's policy deviation, i.e., the distance between the agent's actual policy and the attacker-desired policy. The goal of minimizing the second term $-\beta H(\omega)$ is to maintain similar policy deviations among all agents during the same attack epoch. In this way, all the agents within the population will be poisoned in a synchronous manner. Additionally, β is a weight parameter that controls the importance of the second term, which can be fixed manually or adjusted during the learning of attack strategy.

Algorithm 5 Training of Population-based Attack Strategy

Given
  N proxy agents with natural dynamics models $\{T_0^n\}_{n=1}^N$
  initial policies $\{\pi_0^n\}_{n=1}^N$
  attacker-desired policy π^*
Compute ideal Markovian process $P^* \leftarrow \pi^* \times T_0$
for episode_num = 1, 2, 3 ... **do**
  reset $\{T_0^n\}_{n=1}^N$ and $\{\pi_0^n\}_{n=1}^N$
  attack initial states $x_0^n \leftarrow \langle \pi_0^n, T_0^n \rangle$ where $n \in N$
  for attack epoch i = 1, 2, 3 ... **do**
    attack N proxy agent in *parallel*, $n \in N$
      $u_i^n \leftarrow \sigma(x_{i-1}^n)$ ▷ choose n^{th} attack actions
      $T_i^n \leftarrow T_{i-1}^n(u_i^n)$ ▷ poison n^{th} environment dynamics
      $\pi_i^n \leftarrow f_{victim}(T_i^n, \pi_{i-1}^n)$ ▷ n^{th} agent learns policy
      $P_i^n \leftarrow \pi_i^n \times T_i^n$ ▷ compute n^{th} actual Markovian process
      $c_i^n \leftarrow D_{KLR}(P_i, P^*)$ ▷ compute attack cost of n^{th} agent
      $x_i^n \leftarrow \langle \pi_i^n, T_i^n \rangle$ ▷ update attacker's state
    obtain distribution $q(\omega_i^n)$ where $\omega_i^n \leftarrow \Delta(\pi_i^n || \pi^*)$ ▷ see Eq. 3.6
    compute attack cost $C_i \leftarrow \sum_{n=1}^N (c_i^n) - \beta H(\omega)$ ▷ see Eq. 3.5
    update attack strategy σ using samples $\{x_{i-1}^n, u_i^n, x_i^n, C_i\}_{n=1}^N \in \mathcal{B}$
    if $\{\max_a \pi_i(a|s) == \pi^*(a^*|s) \mid s \in S^*, n \in N\}$ **then**
      attack is done, go to the next attack episode
    end if
  end for
end for

In summary, the attacker is seeking to learn an attack strategy that minimizes the cumulative discounted population-based attack costs, denoted as $\sum_i^\infty \gamma^i C_i$. The

learning of the population-based attack strategy is summarized as Algorithms 5. Here, line $10 \sim 16$ describe the environment-poisoning attacks against each agent within the population in parallel, generating the individual attack costs. The line $17 \sim 18$ measures the population-based attack costs that considers the overall costs over the whole population as well as the distribution of attack performance. Then the population-based cost is used for optimizing the attack strategy at line 19. Such an attack episode will be terminated if all the agents within the population are attacked successfully.

It is important to note that the learning of the population-based attack strategy takes into consideration the overall attack performance of the population (i.e., centralized training), whereas the deployment of the attack strategy focuses on the progress of each individual agent as it learns (i.e., decentralized implementation). Furthermore, the learned attack strategy can be implemented in a larger population due to its transferability (refer to Section 3.4).

3.6 Experiment

In this section, we empirically evaluate the proposed TEPA with the following objectives: (1) to demonstrate the effectiveness and efficiency of TEPA for forcing RL agents to learn the attacker-desired policy; (2) to demonstrate the transferability of the attack strategy we developed; (3) to assess the attack performance against black-box RL agents, whose algorithms or policies are unknown; (4) to verify the efficiency of the attack against a population of independent RL agents. Implementation codes can be found at *https://github.com/JoanaHXU/TEPA*.

3.6.1 Experiment Settings

In this part, we first describe the environment in the victim's task, and then introduce the implementation details in learning algorithms used by the attacker as well as victims. Finally, we define how the attack performance is measured in these experiments.

3.6.1.1 Environment

Robot navigation is one fundamental problem in robotics and commonly used to test RL algorithms [94]. Therefore, we choose to attack the RL agents performing navigation tasks in Grid World environment.

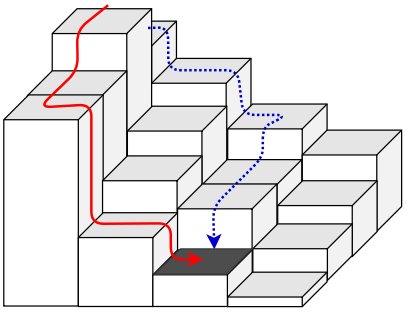


FIGURE 3.4: 3D Grid World. The shallow cell is the goal. The blue line represents the learner's optimal path in the natural elevation setting, whereas the red line is the target path designed by the attacker.

The 3D grid world domain is proposed in [53], to simulate a mountain or rugged terrain. As shown in Figure 3.4, there are associated elevations among cells of the 3D grid world. The success of moving from one cell to the neighbouring cell is proportional to the relative elevation between the two cells. Thus, changing elevation can affect how the environment responds to the agent's action. This means that elevation change is a mechanism to modify the environment transition dynamics. The 3D grid world is the learner's environment. The learner's task is to find an optimal path from the start cell to the goal cell. At each time step, the learner can move in one of the four cardinal directions. The reward is -1 for each step the learner takes until it reaches the goal cell. Here, the attacker's objective is to force the agent to follow a desired path reaching the goal cell. As shown in Figure 3.4, the blue line represents the learner's optimal path in the natural elevation setting, whereas the red line is the target path designed by the attacker.

3.6.1.2 Implementation

Since the attacker continuously manipulates elevations of the 3D Grid World, we adopt Deep Deterministic Policy Gradient (DDPG) [30] when learning the attack strategy. The attack strategy network is represented by a multi-layer neural network as INPUT(80)-FC(400)-ReLU-FC(300)-ReLU-FC(16). Additionally, the

victim agent may choose Q-learning, Sarsa and Monto Carlo (MC) [25] as learning algorithms with parameter configurations: $\epsilon = 0.1, \gamma = 1.0, \alpha = 0.1$.

3.6.1.3 Measurement

The attack is regarded as successful if the victim agent chooses the desired action a^* in target states S^*. We define *attack success rate* to denote how many percentages of the target states set have been attacked successfully. In this experiment, we measure the attack performance using the *attack success rate* during the RL agent's learning process. The efficiency of attack strategy is measured by the convergence rate, which indicates faster converge rates mean better attacking efficiency. Furthermore, a successful transferable strategy is identified as more than 90% success rate for attacking a victim agent, when the attack strategy is learned on a proxy agent.

TABLE 3.1: Description of attack settings: (1) White-box setting where the attacker has full knowledge of the victim agent's learning algorithm and policy model; (2) black-box scenario [A] where the attacker has no idea of the agent's learning algorithm but knows its policy model; (3) black-box scenario [B] where the attacker has no prior knowledge about the victim's learning mechanism. Note that the attacker is assumed to have full knowledge of the victim's training environment.

		White-box	**Black-Box [A]**	**Black-Box [B]**
Agent	Learning Algorithm	✓	✗	✗
	Policy Model	✓	✓	✗
Environment	Dynamics Model	✓	✓	✓
	Stationary State Distribution	✓	✓	✓
	Effect of Hyper-parameters on Dynamics	✓	✓	✓

3.6.2 Evaluations on a White-Box RL Agent

In this part, we focus on evaluating TEPA in the white-box setting where the omniscient attacker has full knowledge of both the victim and the environment, as described in Table 3.1. We first introduce a SOTA baseline, i.e., reward-poisoning approach [15], and then evaluate TEPA performance in comparison with the baseline, showing the effectiveness and efficiency of our approach against a variety of white-box RL agents.

3.6.2.1 Baseline

Our baseline is a SOTA approach, an adaptive reward-poisoning attack [15]. The baseline method shares the same attack objective as ours: to force an RL agent to learn a target policy designed by the attacker. It also formulates the attacker as an RL agent and uses TD3 [31] to learn the attack strategy. Different from our proposed approach, the baseline approach poisons the victim's perceived reward on-the-fly. Specifically, there are two kinds of reward-poisoning methods proposed in [15]. The first one, *Reward Poisoning*, considers the victim's transition and Q tabular $\langle s, a, s', r, Q\rangle$ as the input to the attacker, and accordingly poisons the reward signal r. The second one, *Reward Poisoning with FAA*, combines a fast adaptive attack (FAA) algorithm into the *Reward Poisoning*. Here, according to the distance from the starting state, target states are ranked descendingly and are attacked sequentially. Once the target action is achieved in the target state s^*, the attacker forces the victim to fix the Q value $Q[s^*]$. As such, the *Reward Poisoning with FAA* can control the victim agent's internal learning mechanism.

3.6.2.2 Attack Performance Evaluation

We evaluate the performance of environment-poisoning attack approach in comparison with the SOTA reward-poisoning attack approaches, which is shown in Figure 3.5. The attack performance is measured as attack success rate at each *timestep* during the victim's learning process. In Figure 3.5, we observe a fast-increasing attack success rate comparing with *reward poisoning* and *reward poisoning + FFA*. When the victim's learning process finishes, our environment-poisoning strategy shows nearly 100% attack success rate comparing with 85% *Reward Poisoning with FFA* and 40% *Reward Poisoning*. This result indicates that our environment-poisoning strategy outperforms the reward-poisoning attacks. Note that, comparing with *Reward Poisoning with FFA*, our attack approach poisons the victim's external environment to affect the victim's learning process, without control on the victim, leading to more realistic applications.

Additionally, comparing the original 3D Grid World (as Figure 3.4) with the poisoned one (as Figure 3.6a), we can see that elevations are manipulated within a limited scale, shown as Figure 3.6b. Specifically, the terrain of the target path is manipulated as it descends from the starting location to the destination one, while

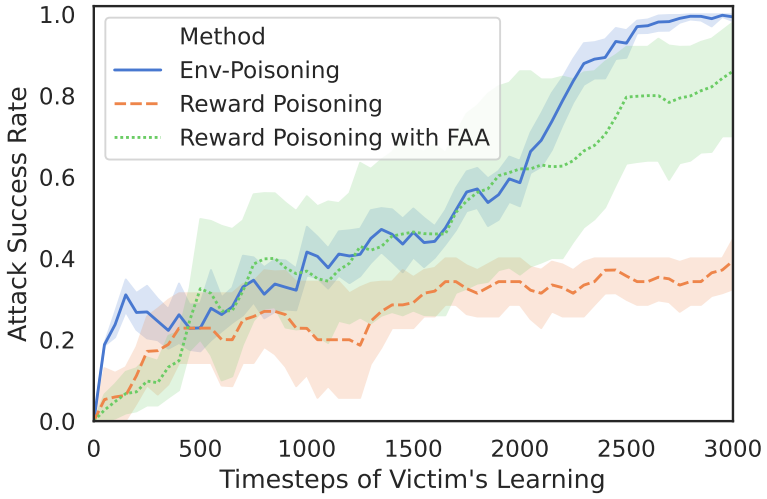
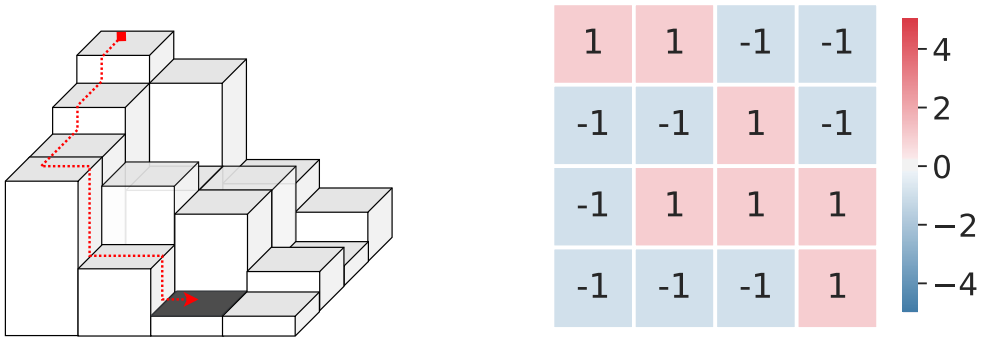


FIGURE 3.5: Attack performance evaluation in *white-box* setting. Environment-poisoning attack outperforms the baseline reward-poisoning approaches in 3D Grid World.

terrains along other paths become rugged. Within such a poisoned environment, the target path has become one optimal choice for the victim, and thereby the victim's policy has been successfully manipulated as the attacker-desired manner. As can be seen, the changes in the environment are explicable and limited.



(A) Poisoned 3D World (B) Changes of Terrain

FIGURE 3.6: Visualization of the poisoned 3D grid world.

It is necessary to note, the size of the RL agent's state space affects the difficulty of learning an adaptive attack strategy. The larger the agent's state space, the larger the attacker's action space are and the sparser the attacker-desired states are. Specifically, with a larger action space, an attacker must conduct more trials of changing the victim's environment in order to determine the most effective attack actions. Also, to force an RL agent to visit all the target states which are sparse

in the environment, the attacker is likely to take more attack actions for greatly changing the agent's training environment.

3.6.2.3 Attack Generalization Evaluation

Most training-time attacks are designed for a specific RL algorithm, such as batch RL [14] and Q learning [15]. To show that our environment-poisoning attack approach threats various types of RL learning algorithms, we evaluate our attack against RL agents with three learning algorithms respectively: Q-learning, Sarsa and MC [25]. Note that, in this experiment, the attack strategy is learned on one kind of RL agent and then be applied on the agent which adopting the same learning algorithms. Figure 3.7 displays the attack success rates at each *timestep* along the victim's learning process. We observe that Sarsa and Q-learning agents are poisoned successfully within 4000 timesteps while MC-agent obtains the target policy within 10000 timesteps. Therefore, our environment poisoning attack is generally effective against a wide variety of RL agents, regardless of the victim's learning algorithm. In spite of successful attacks on various RL agents, the efficiency of an attack depends on how the victim learns its policy.

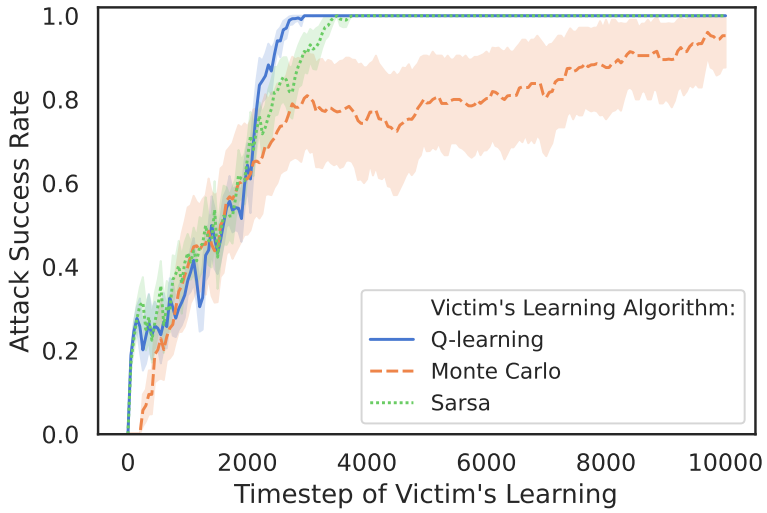


FIGURE 3.7: Attack generalization evaluation in *white-box* setting.

3.6.3 Evaluations on a Black-Box RL Agent

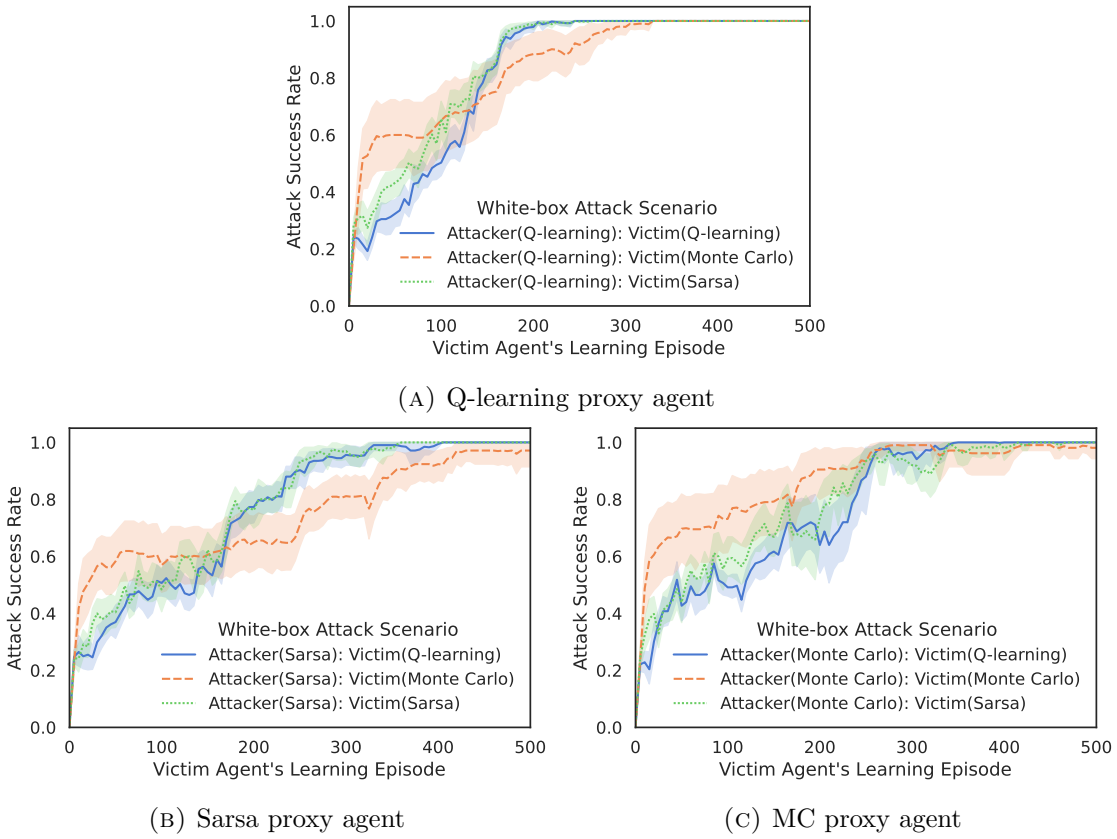
In this part, we evaluate the effectiveness of TEPA against black-box RL agents whose learning algorithms and/or policy model are unknown to the attacker, as shown in Table 3.1. Our proposal is to transfer the learned attack strategy, which is learned on a proxy agent, to poison an unknown RL agent. In this study, we demonstrate empirically the transferability of an environment-poisoning attack strategy. Following that, we evaluate the performance of attacks on a black-box victim agent using the transferred strategy. Here, experiment settings and implementations are the same as those of the white-box attack.

3.6.3.1 Transferability Evaluation

When evaluating the transferability of the environment-poisoning attack strategy, we develop three white-box proxy agents: Q-learning, Sarsa and MC. We learn three attack strategies based on these proxy agents, respectively. Afterward, each attack strategy is transferred to poison victims adopting various learning algorithms.

Figure 3.8 shows performances of each attack strategy against different victims. Note that the attack success rates are measured at each *episode* during the victim's learning process. Figure 3.8 shows that attack success rates reach 90 percent and higher within 500 learning episodes of victims. If observing each plot in Figure 3.8, we find that the attack is more efficiency when the victim and the proxy agent adopt the same learning algorithms, comparing with the transferred attack performance. This result is reasonable as that the strategy is learned based on a specific RL learning algorithm. Nevertheless, the attack performances are promising when the victim and the proxy agent use different algorithms. Experiment results show that each strategy successfully poisons the three kinds of victims' policy, as a result, the transferred attack strategy is generally effective to various RL agents.

Additionally, as shown in Figure 3.8a and 3.8b, the efficiency of the transferred attack against the MC-victim is somewhat lower than that against the Q/Sarsa victim. This result can be explained by the characteristics of the learning algorithms. Here, MC is a on-policy experience-based learning algorithm. Sarsa and Q-learning are temporal-difference (TD) algorithms. TD learns the policy online



(A) Q-learning proxy agent

(B) Sarsa proxy agent

(C) MC proxy agent

FIGURE 3.8: Transferability evaluation of environment-poisoning attack strategy. Transferable attack strategies successfully poison the three types of victims' policies. Proxy agents and victim agents are explicitly different.

at every time step, whereas MC must wait until the end of the episode for the policy to be updated. In practice, it appears that TD learning works more efficiently than MC, which affects the attack efficiency. In summary, the attack strategy can be successfully transferred among agents which use different learning algorithms, while the attack efficiency can also be influenced by the characteristics of the RL learning algorithms.

3.6.3.2 Attack Performance Evaluation

Due to the transferability, we propose that environment-poisoning attack can be applicable to a victim agent of which learning algorithm is unknown to the attacker. Specifically, the attack strategy is learned from a white-box proxy agent and then transferred to attack the black-box victim agent in the same training environment. As the attack should be responsive to the victim's learning progress,

policy modelling is used to determine the victim's policy feature from its trajectory. The policy modelling is conducted by an encoder-decoder network which is described as Table 3.2.

TABLE 3.2: Architecture of Encoder-Decoder network used for representing policy. ReLU and Layer normalization is applied after each layer except the last one.

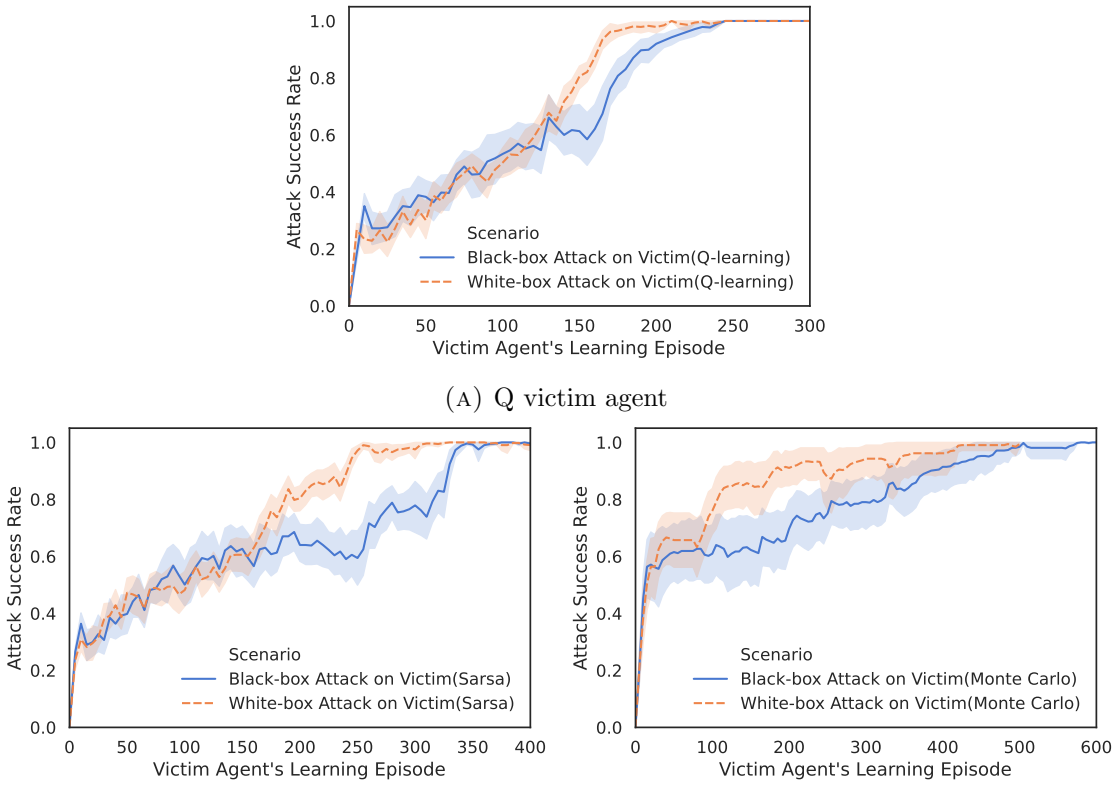
Encoder		
Type	Input Dim.	Output Dim.
Linear	20	36
Linear	36	36
Linear	36	5
Decoder		
Type	Input Dim.	Output Dim.
Linear	6	36
Linear	36	36
Linear	36	4

Figure 3.9 displays the comparison results of the environment-poisoning attack against the white-box victim and the black-box victim. Note that the white-box attack does not involve transferability of the attack strategy. As shown, the performance of a black-box attack is comparable with that of a white-box attack, while its success rate grows slowly. It is partly due to the fact that the attacker needs time to collect the victim's trajectory data and deduce the policy features of the victim. In this simulation scenario, the result indicates that our proposed approach can successfully attack an RL agent whose internal learning mechanism is unknown to the attacker.

3.6.4 Evaluations of on a Population of RL Agents

In this part, we evaluate the proposed TEPA against a population of independent RL agents, seeking to manipulate the policies of all the agents synchronously.

The attack strategy is learned following the population-based attack cost (see Equation 3.5), based on a set of proxy agents which adopts Q-learning, Sarsa and MC as learning algorithms. The size of the proxy population is three. Then the strategy is applied to poisoning a population of fifteen victim agents that adopt different learning algorithms (i.e., Q-learning, Sarsa, MC). In these experiments, the proportion of each kind of agent is $1 : 1 : 1$.



(A) Q victim agent

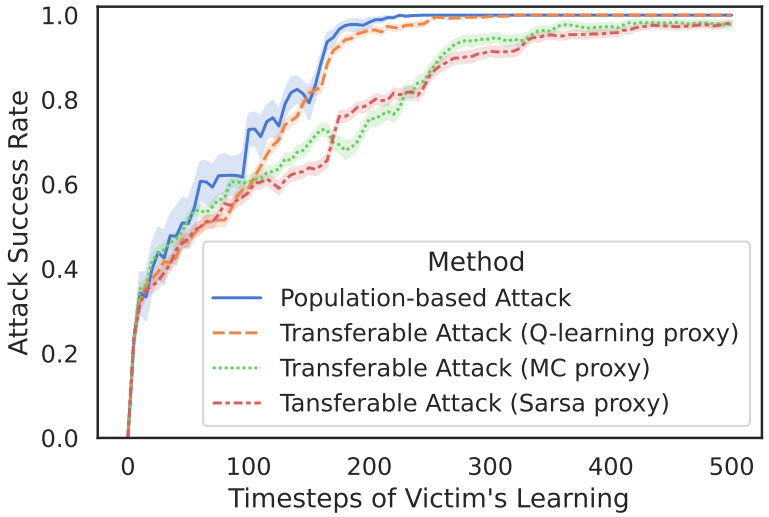
(B) Sarsa victim agent

(C) MC victim agent

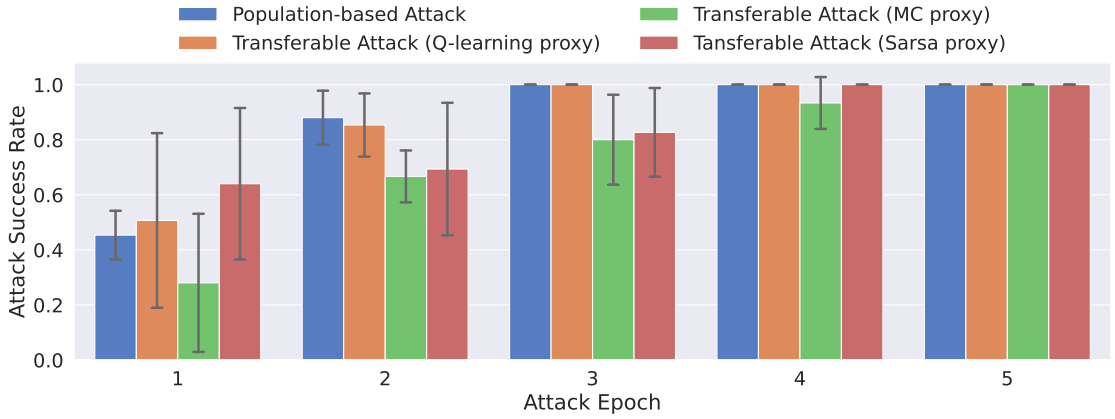
FIGURE 3.9: Attack performance evaluation in *black-box* setting. An attack strategy, learned on a white-box proxy agent, successfully attack black-box victim agents.

To evaluate the effectiveness and efficiency of the proposed synchronous population-based attack, we compare it with a transferable attack strategy which is learned based on a single proxy agent and can be transferred to poison various independent RL agents. The performance of their attacks is evaluated in 50 runs of attack deployment and the results are presented in Figure 3.10a. Figure 3.10a illustrates the attack performance over a population (including 15 victim agents) over time. We also present a figure that shows the mean and variance of attack success rates over a population along attack epochs within a single attack deployment. This can be found at Figure 3.10b.

As shown in Figure 3.10, the proposed population-based attack is more effective than transferable attack strategies learned on MC and Sarsa proxy agents since it is capable of poisoning the behaviour of the entire population within three attack epochs. Similar attack efficiency is achieved as the baseline, transferable attack strategy learned on a Q proxy agent, which has been empirically demonstrated



(A) Attack performance along timesteps of victim's learning.



(B) Attack performance along epochs of attacker's poisoning.

FIGURE 3.10: Evaluation of environment-poisoning attack on a population of RL agents.

to be more transferrable. Nevertheless, the variance of its attack performance relative to the population is considerably lower than that of the baseline (i.e., Q-proxy one), especially at the first attack epoch, as shown in Figure 3.10b. The lower variance of attack performance means that the poisoning behaviour of a population is more synchronous. In this way, there is no extremely abnormal agent within a population, resulting in a low probability for being detected.

In conclusion, experimental results demonstrated that the population-based attack is capable of poisoning the behaviour of a population of RL agents efficiently (i.e., fewer attack epochs) and synchronously (i.e., less variance in performance at each attack epoch). Note that, even though the population-based attack strategy

is learned from a group of three proxy agents, it can be applied to poisoning a population of more agents due to its transferability property.

3.7 Summary

In this chapter, we propose an environment-poisoning attack against an RL agent at training time, seeking to force the RL agent to learn an attacker-desired policy via minimized changes in environment dynamics. We design the attack framework as a bi-level MDP framework and define the attack optimality criterion for learning an adaptive attack strategy. Our analysis shows the attack strategy is transferable, enabling the effective poisoning on a black-box RL agent. Furthermore, we investigate the environment-poisoning attack on a population of independent RL agents to induce all the agents to follow the same attacker-desired behaviours.

Experiments are conducted to evaluate the environment-poisoning attack against white-box and black-box agents as well as a population of RL agents. We first show the effectiveness and efficiency of our proposed attack in comparison with SOTA reward-poisoning attacks in white-box settings. We then verify the transferability, showing that attack strategies can be trained using a white-box proxy agent and transferred to poison different types of RL agents. Also, we analyze how the proxy agent affects the transferable attack strategy, and conclude that more exploration by the proxy agent can lead to an attack strategy with better transferability. Moreover, we combine the generative policy representation with environment-poisoning attacks, and demonstrate our work is effective for attacking a black-box RL agent whose both learning algorithms and policy are unknown. Finally, we evaluate a population-based attack strategy which achieves to efficiently and synchronously poison behaviours of a population of independent agents.

This work automatically generates an adaptive attack on training environments for poisoning RL policies. While it yields promising results, the simulation environment adopted in the work is quite simplistic. An investigation of more sophisticated environments, in which the environment transition-dynamics function is difficult to be known by the attacker, is a necessary research topic.

Chapter 4

Environment Poisoning with Minimal Prior Knowledge

4.1 Introduction

The transferable environment-poisoning attack (TEPA) succeeds in poisoning the policy of an RL agent even if the agent's learning algorithm and policy are unknown by the attacker, i.e., achieves training-time attacks against a black-box RL agent. Nevertheless, the full knowledge of the training environment lies at the heart of TEPA as well. As a result, TEPA is confined to white-box environments and poses limited threat to complex RL systems since environment-dynamics information is generally private or unknown in most real-world applications. We argue that the prior knowledge of environment makes an attack approach unrealistic. To alleviate such a limitation, we develop a novel attack approach that is independent from internal information of an RL system: a) how the RL agent learns its policy, i.e., the learning algorithm and the policy model; b) how the environment responds to the agent's action, i.e., the environment dynamics; c) how hyper-parameters determine the environment dynamics, i.e., the environment causal mechanism.

Specifically, we propose a Double-Black-Box Environment Poisoning Attack (DBB-EPA) approach, which achieves policy compulsion on an *unknown* RL agent in an *unknown* environment. To the best of our knowledge, this is the first environment-poisoning attack that does not rely on prior knowledge of both the agent's learning mechanism (i.e., policy training algorithm and policy model structure/parameters)

and its environment model (i.e., transition and reward functions). Assuming only the ability to alter the environment hyper-parameters, our attack aims to force a black-box RL agent to learn an attacker-desired policy, with minimal and adaptive environment poisoning. To this end, we first investigate how to infer the internal information of an RL system, and then learn an adaptive attack strategy based on the approximation of our attack objective. Specifically, given observations of an RL agent's trajectories during its learning process, we jointly train: a) an Encoder-Dual-Decoder network that learns a low-dimensional latent representation of the RL system's internal information; b) an attack strategy, conditioned on the latent representation and environment hyper-parameters, that manipulates the RL agent's policy using minimal environment poisoning. In contrast to the existing EPAs [16, 46, 95] that are only effective in discrete state domains, our work provides a tractable approach to attack deep-RL (DRL) agents in continuous environments.

In summary, the contributions of this chapter are as follows [1]:

- We propose an environment-poisoning attack approach in the double-black-box setting, where both the RL agent's learning mechanism and its environment model are unknown to the attacker – DBB-EPA.
- We design an Encoder-Dual-Decoder network to infer internal information of a black-box RL system and construct an adaptive attack strategy conditioned on the resultant latent representation by approximating our attack objective.
- We show that DBB-EPA achieves performance comparable to the white-box attack (i.e., TEPA) on a navigation task in the grid world. We further evaluate our attack against a DRL agent on a control task in continuous domains, showing the feasibility and the scalability of DBB-EPA on more complex RL systems.

4.2 Problem Statement

This section describes the attack framework proposed for environment poisoning. We first provide notations and preliminaries of the attack framework, followed by the mathematical description of our attack problem in a double-black-box setting.

[1] The work in this chapter has been published in Proceedings of the 21th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'22).

4.2.1 Notations and Preliminaries

We adopt the bi-level MDP architecture which has been illustrated in Figure 3.1. The task of poisoning an RL agent's (i.e., victim) policy is performed by another RL agent (i.e., the attacker) that operates on a different timescale from the victim. Specifically, with a particular attack frequency, the attacker manipulates the victim's training-environment hyper-parameters in response to the victim's learning progress. The attacker's objective is to achieve an optimal attack strategy that succeeds in poisoning the victim's policy while minimizing changes to the victim's environment. The attack framework is formally described as follows.

Victim-level MDP. The Markovian environment of a victim is denoted by the tuple $\langle S, A, T_e, r, d_0, \hat{\gamma} \rangle$. Here, $s \in S$ is an environment state and $a \in A$ is the victim's action. $T_e(s'|s, a)$ represents the victim's dynamics function that tells the victim's state transition probabilities from s to s', given the action a and the environment hyper-parameter e. $r : S \times A \times S \to \mathbb{R}$ represents the victim's reward function. $d_0(s)$ is a distribution over the victim's initial states and $\hat{\gamma}$ is a discount factor. The victim aims to learn an optimal policy $\pi(a|s)$ that maximizes the cumulative discounted rewards $\sum_t^{\infty} \hat{\gamma}^t r_t$. Its trajectory τ is a sequence of state-action pairs generated by π. Additionally, the attacker-desired policy is represented as π^* and the corresponding desired trajectory is denoted as τ^*.

Attacker-level MDP. An attacker, which is regarded as an outer-loop RL agent as shown in Figure 3.1, treats the victim-level system as its dynamics environment. We describe the attacker's Markovian process by the tuple $\langle X, U, F, c, \gamma \rangle$ as follows.

- X is the attacker's state space. $x_i \in X$ represents an attack state that contains information about the victim's behaviour policy and its environment dynamics at the i^{th} attack epoch.
- U is the attacker's action space. An attack action $u \in U$ represents a manipulation on the victim's environment hyper-parameter e. Here, e influences how the training environment responds to the victim's action, i.e., environment dynamics. e_i represents the poisoned environment which has been changed by aggregate attack actions $\{u_1, u_2, ..., u_i\}$. e_0 denotes the victim's natural environment.

- $F : X \times U \times X \to [0, 1]$ is the attacker's probabilistic state transition function. It captures how the victim updates its policy under effect of the attacker's action. Specifically, $F(\pi'|\pi, u)$ represents the probability that the victim updates its policy from π to π' in the environment changed by u.
- $c : X \times U \times X \to \mathbb{R}$ is a function denoting the attack cost. It jointly reflects the attack performance (i.e., difference between the victim's policy and the attacker-desired one) and the attack efforts (i.e., aggregated changes to the victim's environment).
- γ is a discount factor.

Note that we use i to represent the index of an attack epoch in which the victim learns its policy for some episodes.

4.2.2 Attack Problem Formulation

The attacker aims to learn an attack strategy $\sigma(u|x)$ that achieves policy compulsion on the victim via minimal changes to the victim's training environment. In detail, the attack objective is to minimize the deviation between the victim's policy and the attacker-desired one, while at the same time minimizing the deviation between the poisoned environment dynamics and the natural ones. Therefore, the attack optimization problem is to minimize the cumulative attack costs, which is denoted as:

$$\begin{aligned} &\min_{\sigma} \sum_{i=1}^{\infty} \gamma^i c_i \\ &s.t. \\ &c_i := \Delta(P_i(s', a'|s, a)||P^*(s', a'|s, a)) \\ &P_i(s', a'|s, a) = T_{e_i}(s'|s, a)\pi_i(a'|s') \\ &P^*(s', a'|s, a) = T_{e_0}(s'|s, a)\pi^*(a'|s'), \end{aligned} \tag{4.1}$$

where c_i is the attack cost at the i^{th} attack epoch. P_i is a stochastic process [53] over state-action pairs, where the victim follows the policy $\pi_i(a|s)$ in the environment e_i which has been modified by a sequence of tweaks $u_{1:i}$. Similarly, P^* represents an ideal stochastic process, where the victim adopts the attacker-desired policy $\pi^*(a|s)$ in the natural environment e_0. Referring to the definition of stochastic processes

P_i and P^*, $\Delta(P_i||P^*)$ measures the deviation jointly caused by the victim's actual policy π_i and its underlying environment dynamics T_{e_i}. We can see that $\Delta(P_i||P^*)$ mathematically describes the attack cost.

Unfortunately, solving the attack optimization problem is intractable in complex or real-world RL systems due to challenges in computing $\Delta(P_i||P^*)$. These challenges are mainly caused by (1) the RL agent's learning algorithm and policy model π are generally black-box to the attacker in the real world; (2) the environment-dynamics model T is typically unknown in most RL tasks, such as control tasks in continuous state domains; (3) even though the environment hyper-parameters can be accessed, it is hopeless to obtain prior knowledge about how the environment hyper-parameters e determine the transition dynamics T_e.

In the following, we propose an approach that simultaneously learns how to infer internal information of an RL system, how to approximate our attack objective, and how to learn an optimal strategy addressing the attack optimization problem. We make minimal assumptions on the attacker's prior knowledge, resulting in a more realistic and scalable attack approach to complex RL tasks.

4.3 Methodology

This section presents our attack approach and introduces how we tackle the challenges caused by the limited prior knowledge. We start by describing the attack procedure in the double-black-box setting. We then consider how to infer the internal information of the RL system, and finally learn an adaptive environment-poisoning strategy based on an approximation of our attack objective.

4.3.1 Attack Procedure

To lure an RL agent's policy learning into a desired direction, the attacker poisons the training-environment hyper-parameters according to the information about the agent's behaviour policy and environment dynamics. Unfortunately, such information is hidden from the attacker, resulting in challenges for designing and deploying

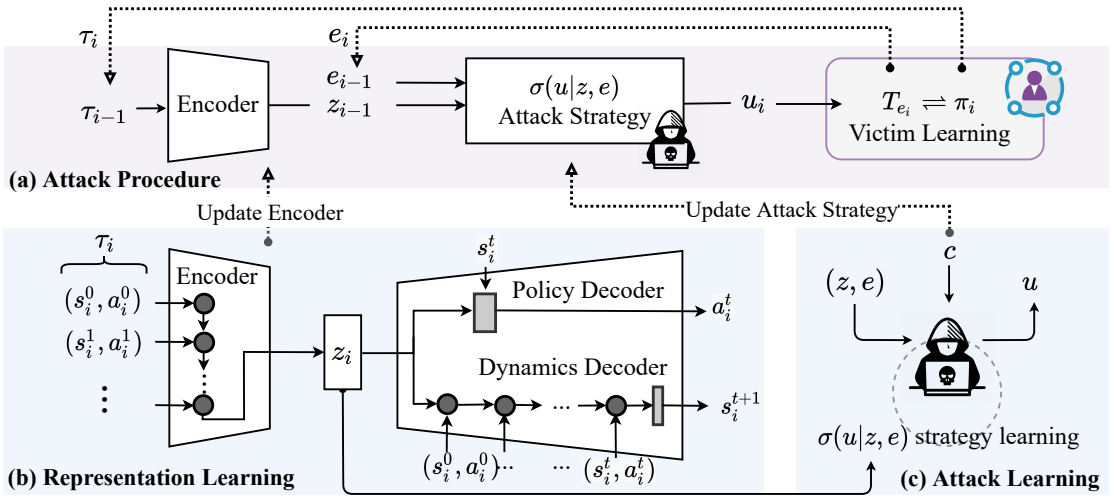


FIGURE 4.1: Illustration of double-black-box environment-poisoning attacks: (a) shows the attack procedure; (b) and (c) describe the latent representation learning and attack strategy learning, respectively. The solid line denotes data transfer and the dotted line represents data update.

an adaptive attack. To learn an adaptive environment-poisoning strategy, we incorporate an inference module into the attack framework, which can infer the features of the victim's behaviour and environment during its learning progress.

As shown in the part (a) of Figure 4.1, an encoder and an attack strategy are essential components of the attack procedure. The encoder captures the feature of the victim's learning progress, and the attack strategy poisons the victim's environment based on the inferred feature. In detail, the attack procedure consists of two stages at each attack epoch i. First, given an observation of the victim's trajectories, the attacker uses the encoder to infer a joint feature z_{i-1} of the victim's policy and its training environment. Second, conditioning on the inferred feature z_{i-1}, the attacker conducts a poisoning action u_i on the environment hyper-parameter e_{i-1}, resulting in poisoned dynamics T_{e_i}. Thereby, the victim, which explores in the training environment T_{e_i}, evaluates and adjusts its policy π_i according to the feedback from the poisoned environment. Such an attack procedure continues until the victim acquires the attacker-desired policy.

4.3.2 Attack Learning

As the encoder and the strategy are learned simultaneously, the learning of the double-black-box attack is composed by 1) latent representation learning and 2)

attack strategy learning, as shown in the part (b) and the part (c) of Figure 4.1.

4.3.2.1 Latent Representation Learning

Without access to the internal information of the RL victim's system, the attacker can only observe the victim's trajectory $\tau = \{s^0, a^0, s^1, a^1, \dots s^t, a^t\}$. Based on the trajectory τ_i collected at the i^{th} attack epoch, the attacker learns a latent embedding z_i that best represents the victim's stochastic process $P_i(s', a'|s, a) = T_{e_i}(s'|s, a) \times \pi_i(a'|s')$. Similarly, τ^* refers to the attacker-desired trajectory, which is used to learn z^* that is the latent representation of the desired stochastic process $P^*(s', a'|s, a) = T_{e_0}(s'|s, a) \times \pi^*(a'|s')$. Note that z reflects the joint information of the victim's environment dynamics and its behaviour policy. As for the effect of environment hyper-parameters (i.e., causal mechanism), it is implicitly expressed in the dynamics.

To learn the latent representation, we design an Encoder-Dual-Decoder network consisting of one encoder network $\mathcal{E}$ and two decoder networks $\{\mathcal{D}^\pi, \mathcal{D}^T\}$, shown as the part (b) of Figure 4.1. Specifically, the encoder $\mathcal{E}$ approximates $f_\eta(z_i|\tau_i)$ which learns a latent embedding z_i based on the victim's trajectory τ_i. $\mathcal{E}$ is designed as a Long Short Term Memory (LSTM) network to learn long-term dependencies in the trajectory. To learn the encoder $\mathcal{E}$, we interpret the embedding z_i via two decoders: a policy decoder $\mathcal{D}^\pi$ and a dynamics decoder $\mathcal{D}^T$. Here, the policy decoder $\mathcal{D}^\pi$ is a multilayer perceptron (MLP) network. $\mathcal{D}^\pi$ learns $f_\theta(a_i^t|s_i^t, z_i)$ which maps the victim's state s_i^t and the embedding z_i to the distribution over the victim's actions a_i^t. And the dynamics decoder $\mathcal{D}^T$ is a LSTM network that approximates $f_\phi(s_i^{t+1}|s_i^t, a_i^t, z_i)$. It predicts the next state s_i^{t+1} on the condition of the embedding z_i and the state-action pair $\{s_i^t, a_i^t\}$.

Given a collection of the victim's trajectories, we learn the parameter η of the encoder $\mathcal{E}$ and parameters θ, ϕ of the dual decoder $\{\mathcal{D}^\pi, \mathcal{D}^T\}$, by maximizing the following negative cross-entropy objective:

$$\mathbb{E}_{\tau_1 \sim \mathcal{T}, \tau_2 \sim \mathcal{T} \setminus \tau_1} \left[\sum_{\langle s, a, s', a' \rangle \sim \tau_2} \log f_\theta(a'|s', f_\eta(\tau_1)) + \log f_\phi(s'|s, a, f_\eta(\tau_1)) \right], \tag{4.2}$$

where $\mathcal{T}$ is a collection of trajectories at one attack epoch. τ_1 and τ_2 are two different trajectories from the collection $\mathcal{T}$, or from the set $\mathcal{T}^*$ of attacker-desired trajectories.

The encoder $\mathcal{E}$ is particularly important in this work, which is used to produce a latent embedding z given a sequence of state-action pairs. The property of the latent representation makes our attack approach independent of the environment type, i.e., our DBB-EPA can be generally applied to both discrete and continuous environments.

4.3.2.2 Attack Strategy Learning:

Referring to Section 4.2.2, $\Delta(P_i||P^*)$ mathematically defines our attack cost, and its computation is the key challenge in learning an optimal attack strategy within the double-black-box setting. To solve the challenge, we design an approximation of the attack cost in this section.

Depending on the encoder $\mathcal{E}$, the victim's stochastic process $P_i(s', a'|s, a)$ and the ideal one $P^*(s', a'|s, a)$ can be represented by the latent embedding z_i and z^*, respectively. Thus, we approximate $\Delta(P_i||P^*)$ as $\Delta(z_i||z^*)$, and use the Cosine Similarity [96] to measure the distance between z_i and z^* in the latent space:

$$\begin{aligned} \Delta(P_i||P^*) &:= \Delta(z_i||z^*) \\ &= 1 - \frac{z_i \cdot z^*}{\|z_i\|\|z^*\|}. \end{aligned} \tag{4.3}$$

Since z_i is inferred from the victim's experienced trajectories, $\Delta(z_i||z^*)$ only captures the environment changes that have affected the victim's behaviour, rather than measures the aggregate changes across the entire environment (i.e., attack effort). Considering minimizing the attack effort, we measure the aggregate environment changes $\Delta(e_i||e_0)$ using the normalized Euclidean distance between the environment hyper-parameter e_i and the natural one e_0.

Thereby, the approximation of the attack cost c_i is a combination of $\Delta(z_i||z^*)$ and $\Delta(e_i||e_0)$, which is denoted as:

$$\begin{aligned} c_i &= (1-\omega) \times \Delta(z_i||z^*) + \omega \times \Delta(e_i||e_0) \\ &= (1-\omega) \times (1 - \frac{z_i \cdot z^*}{\|z_i\|\|z^*\|}) + \omega \times \frac{\|e_i - e_0\|_2}{\|e_{limit} - e_0\|_2}, \end{aligned} \tag{4.4}$$

where $\omega \in [0, 1]$ is the weight parameter and e_{limit} indicates the limit value of environment hyper-parameters. In summary, DBB-EPA uses an encoder to obtain latent representations of an RL system's internal information, and learns an adaptive attack strategy based on an approximated attack cost. The learning of DBB-EPA strategy is finalized as Algorithm 6. Due to the representation and the approximation in the latent space, DBB-EPA is applicable in poisoning an RL agent in both discrete and continuous environments, as empirically discussed in the next section.

Algorithm 6 Learning of DBB-EPA Strategy

for n_episode = 1,2,... **do**
  Reset victim's embedding z_0 and environment e_0
  for attack epoch i=1,2,... **do**
    $u_i \leftarrow \sigma(z_{i-1}, e_{i-1})$ ▷ choose attack action
    $T_{e_i} \leftarrow T_{e_{i-1}}(u_i)$ ▷ alter environment hyper-parameters
    observe trajectory τ_i when victim learns π_i in T_{e_i}
    update encoder $\mathcal{E}$ and dual-decoder $\mathcal{D}^\pi$, $\mathcal{D}^T$ with τ_i
    $z_i \leftarrow \mathcal{E}(\tau_i)$, $z^* \leftarrow \mathcal{E}(\tau^*)$ ▷ infer latent representations
    $c_i \leftarrow (1-\omega) \times \Delta(z_i||z^*) + \omega \times \Delta(e_i||e_0)$ ▷ approximate attack cost
    save transition $\langle \{z_{i-1}, e_{i-1}\}, u_i, \{z_i, e_i\}, c_i \rangle$ into replay buffer $\mathcal{B}$
    update attack strategy $\sigma(u|z, e)$ using samples in $\mathcal{B}$
    if $\tau_i == \tau_*$ **then**
      attack is done, go to the next episode
    end if
  end for
end for

4.4 Experiment

In this section, we first evaluate DBB-EPA on a tabular-RL agent in a didactic grid-world task, showing that DBB-EPA achieves effective training-time attack with minimal prior knowledge of the RL system. Then, we show the feasibility and

the scalability of DBB-EPA in more complex RL tasks. Implementation codes can be found at *https://github.com/JoanaHXU/DBB-EPA*.

4.4.1 Discrete State Domains

We evaluate DBB-EPA on a tabular-RL agent performing a navigation task in a 3D grid world [53, 95] as shown in Figure 4.2, where the environment-dynamics model is unknown. The purpose of this experiment is to evaluate the design of the inference module (as the part (b) in Figure 4.1), and to discuss the approximation of the attack cost (as Equation 4.4).

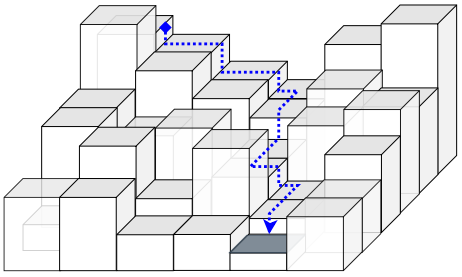


FIGURE 4.2: Discrete state domains: 3D grid world (6x6).

4.4.1.1 Experiment Setting

Implementation and Measurement. We adopt Deep Deterministic Policy Gradient (DDPG) [30] to learn the attack strategy. We measure the attack performance using *attack success rate* [95] at each episode during the victim's learning process. Here, *Attack success rate* is the percentage of the attacker-desired states that have been attack successfully, specifically, successful attack is the one in which the victim performs the attacker-desired action in the desired state.

4.4.1.2 Results and Discussion

Attack Performance Evaluation. We evaluate DBB-EPA performances in comparison with the white-box attack (i.e., TEPA) [95]. All the results are generated by the attack strategies which are learned and evaluated on a Q-learning RL agent. As shown in Figure 4.3, attack success rate of DBB-EPA is comparable to

that of the white-box attack. Consequently, this result suggests that our proposed attack is capable of poisoning a black-box RL agent's policy in an unknown environment. The result further indicates the effectiveness of the proposed inference module and the designed attack-cost approximation.

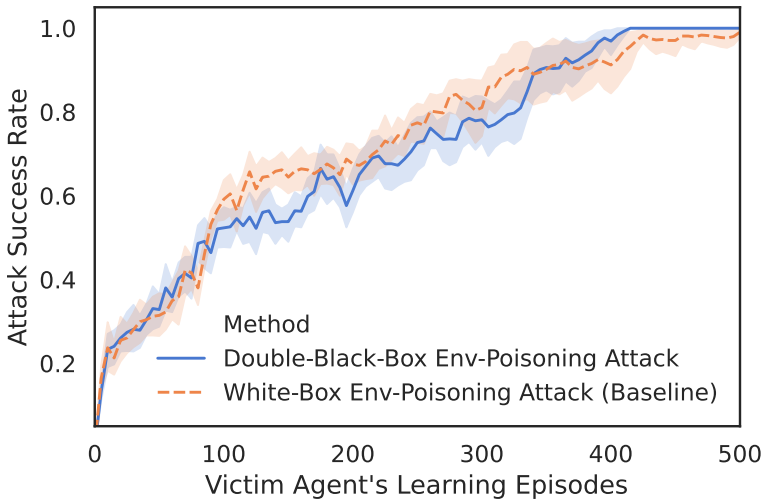


FIGURE 4.3: Performance of double-black-box attack in comparison with white-box attack in 3D Grid World.

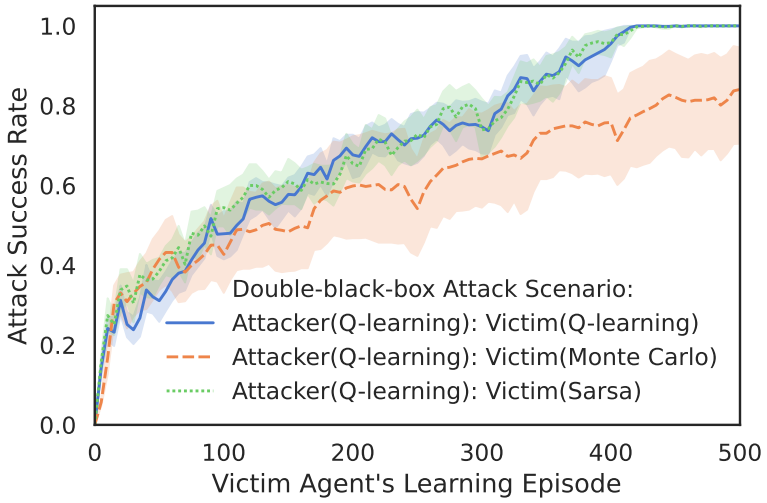


FIGURE 4.4: Performance of double-black-box attack against different RL agents in 3D Grid World.

Transferability Evaluation. To evaluate the transferability of the DBB-EPA strategy, we learn an attack strategy based on a black-box Q-learning proxy agent, and deploy the strategy on three kinds of victim agents of which learning algorithms

include Q-learning, Sarsa and Monte Carlo [25]. As shown in Figure 4.4, the Q-learning and Sarsa victims obtain the attacker-desired policy within 500 learning episodes; while MC victim reaches 80% success rate at 500^{th} episode and needs more time to acquire the attacker-desired policy. We see that the efficiency of the transferred attack against the MC-victim is somewhat lower than that against the Q/Sarsa victim. This result can be explained by the characteristics of victims' learning algorithms. Specifically, Sarsa and Q-learning are temporal-difference (TD) algorithms while MC is an on-policy experience-based learning algorithm. TD learns the policy online at every time step, whereas MC has to wait until the end of the episode for the policy to be updated. In practice, it appears that TD learning works more efficiently than MC, which affects the efficiency of learning the attacker-desired policy. In summary, our DBB-EPA strategy can be successfully transferred among agents which use different learning algorithms, while the attack efficiency is influenced by the characteristics of the victim's RL learning algorithm.

Analysis of Weight Parameter ω. When designing the attack-cost approximation as Equation 4.4, we explicitly consider the aggregate environment changes with a weight parameter ω. Here, we empirically show the effect of ω on the learned attack strategy. Table 4.1 displays deviations between the poisoned environment and the natural one, when the attack success rate has reached 100%. Deviations of environment hyper-parameters are measured by the L_2 norm, and they are also represented by the percentage of hyper-parameters that have been altered. As shown in Table 4.1, the environment deviation decreases as ω increases. Consequently, the explicit representation of environmental deviations contributes positively to the control of attack efforts. However, according to our observation, it is becoming increasingly difficult to train a successful attack strategy with the increasing ω. For example, the training with $\omega = 0.1$ converges within 300 episodes while the training with $\omega = 0.5$ takes 800 episodes to converge. Therefore, learning an attack strategy involves a trade-off. The attack strategy which requires fewer attack efforts (i.e. changes in the environment) is more difficult to learn.

Discussions on Computational Time. We present the computational time introduced by the inference module (i.e., Encoder-Dual-Decoder network) on GPU (i.e., Nvidia RTX A6000) in terms of the attack learning and deployment. First, the learning time of the DBB-EPA strategy is 83,066 seconds, which is 1.19 times

TABLE 4.1: Deviations between manipulated environment hyper-parameters with the natural ones, under different settings of weight parameters. Deviation is measured when attack success rate reaches 100%.

ω	Deviation (L_2)	Deviation Percentage
0.0	21.55 ± 1.77	0.538 ± 0.044
0.1	17.88 ± 1.82	0.447 ± 0.045
0.2	17.61 ± 1.90	0.440 ± 0.047
0.3	16.49 ± 1.29	0.412 ± 0.032
0.4	16.43 ± 1.08	0.411 ± 0.027
0.5	16.11 ± 2.02	0.402 ± 0.050

more than that of the white-box TEPA strategy (i.e., 69,537 seconds). The 20% increase is acceptable for learning the internal information of RL systems. Second, the deployment time of the DBB-EPA attack and the white-box TEPA attack are 30.09 seconds and 29.53 seconds, respectively. The impact of Encoder on the deployment time can be negligible (about 2% increase) due to its small network size and small computation load.

4.4.2 Continuous State Domains

Experimental results in the discrete environment have demonstrated the effectiveness of our inference module and attack-cost approximation. In this part, we further evaluate the feasibility and the scalability of DBB-EPA in more complex RL tasks. Note that the complexity of RL tasks is interpreted from two perspectives: (1) the complexity of the RL agent is scaled by shifting from tabular-RL algorithms up to deep-RL algorithms; (2) the complexity of the environment is scaled from discrete state domains up to continuous domains. In the following, we show the DBB-EPA performance against DRL agents in LunarLander.

4.4.2.1 Experiment Settings

LunarLander [97] is a simulation environment for the trajectory optimization problem, where the environment state is continuous and the dynamics model is unknown. As shown in Figure 4.5, the task is to safely land a spaceship between the flags smoothly. The agent (i.e., spaceship) aims to learn an optimal policy which encourages it to lower the distance to the landing pad, decrease its speed to land

smoothly, keep its angular speed at minimum to prevent rolling, and not to take off again after landing. Thus, the agent's reward is determined by its location, velocity and angle. The agent's state space is continuous, and its discrete action space includes doing nothing, firing left orientation engine, firing main engine and firing right orientation engine.

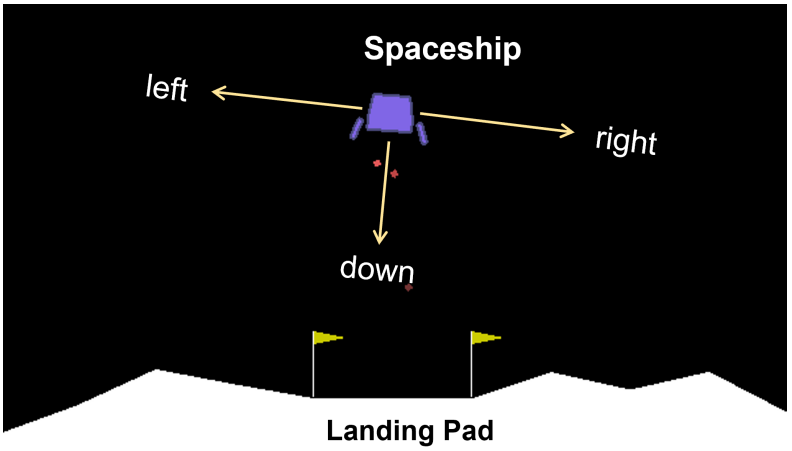


FIGURE 4.5: Continuous state domains: LunarLander.

Attack Settings. In light of the observation that there is random wind which might influence the spaceship's trajectory, we choose *wind* as the environment hyper-parameter that can be implemented by the attacker. Specifically, the wind power affects the spaceship's moving distance, and the wind direction influences the spaceship's moving direction. For example, the spaceship is increasingly difficult to moving downward with increasing upward wind power. Thus, the *wind* implementation is a mechanism to affect the environment dynamics. Additionally, the attacker aims to prevent the spaceship from landing on the pad while keep the spaceship safe. In detail, the attacker-desired trajectory is that the spaceship hovers around the right flag for at least 1000 time steps. We obtain such an attacker-desired policy using the reward-shaping solution [86].

Implementation and Measurement. We adopt Twin Delayed DDPG (TD3) [31] to learn the DBB-EPA strategy. The attack strategy is evaluated on black-box DRL agents of which the learning algorithms include an off-policy algorithm Deep Q-Network (DQN) [28], on-policy algorithms Vanilla Policy Gradient (VPG) [25]

and Proximal Policy Optimization (PPO) [29]. Additionally, the attack performance is measured by the trajectory similarity, i.e., the Cosine Similarity between the latent representations. If the trajectory similarity is more than 90%, we consider that the victim's policy has been successfully poisoned.

4.4.2.2 Results and Discussion

Attack Performance Evaluation. We learn an attack strategy based on a DQN proxy agent and a PPO proxy agent, respectively. The attack strategy is deployed at every 80 episodes during the victim's learning process, and the attack performance is measured at the end of each learning episode. As shown in Figure 4.6, the similarity between the DQN victim's actual trajectory with the attacker-desired one reaches nearly 100% within two attack epochs. It indicates that the DQN victim's policy have been poisoned successfully by the attack strategy which is learned on the DQN proxy. When attacking PPO victim using the strategy learned on the PPO proxy, Figure 4.7 shows that trajectory similarity reaches above 90% within three attack epochs. These results indicate that DBB-EPA is successful in forcing an DRL agent to learn the attacker-desired trajectory in a continuous environment, without prior knowledge of both the agent's algorithm and the training environment.

In more detail, we observe that, in Figure 4.6, the trajectory similarity is decreased at the end of the first attack epoch. This temporary drop means that the agent's policy gradually deviates from the attacker-desired one along with its learning progress. It is time for the attacker to further poison the agent's training environment based on the agent's policy feature and the environment status, to induce the agent to learn the policy towards the attacker-desired direction. Such an observation implies the necessity of a sequential and adaptive attack for manipulating the agent's policy at training time.

Transferability Evaluation. We evaluate the transferability of DBB-EPA strategy in the LunarLander setting. The attack strategies are trained based on a DQN proxy agent and a PPO proxy agent, respectively. Afterwards, they are applied to attack victims adopting different learning algorithms. Figure 4.8 and Figure 4.9 show the performance of the transferred attack strategies. Overall, the DBB-EPA

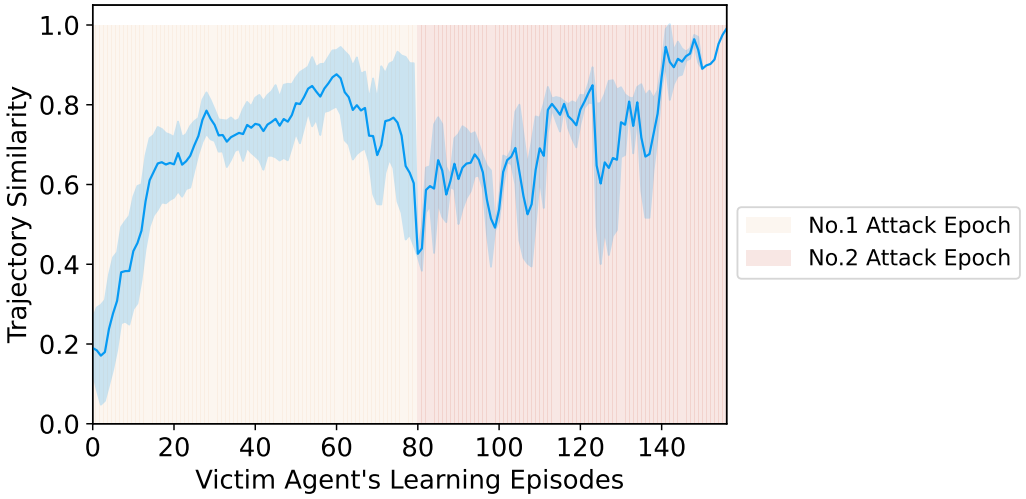


FIGURE 4.6: Attack evaluation on a DQN victim in LunarLaner.

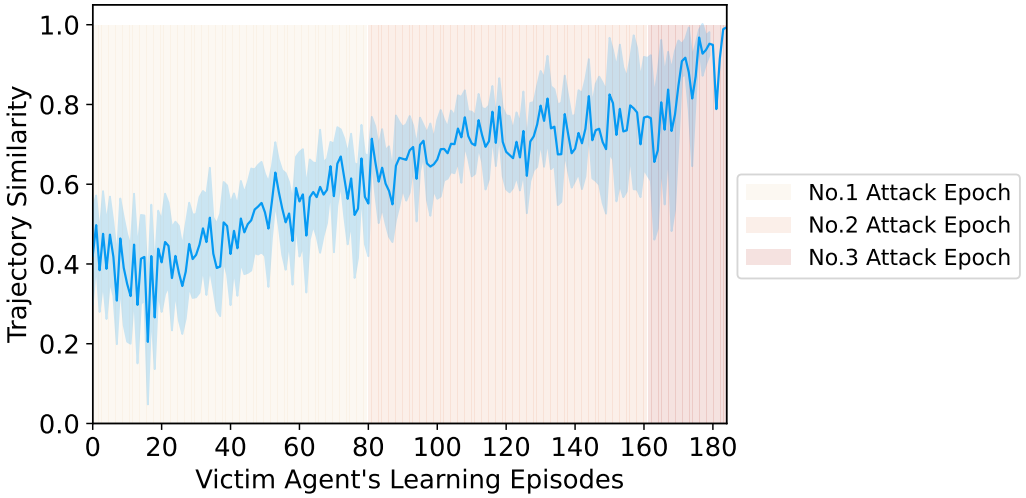
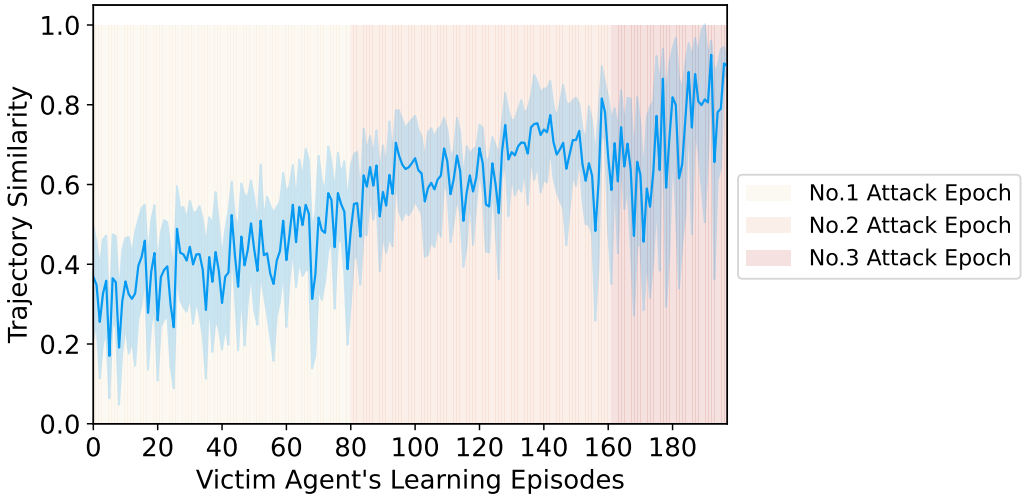


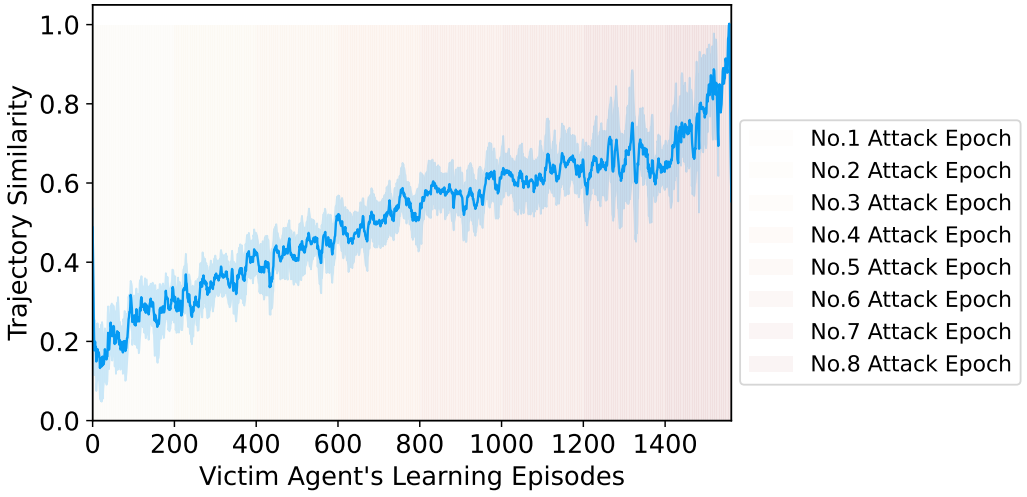
FIGURE 4.7: Attack evaluation on a PPO victim in LunarLaner.

strategies successfully induce the DRL victim to learn the attacker-desired trajectory regardless of the victim's learning algorithm. In detail, as shown in Figure 4.8, the PPO victim's policy is poisoned within 3 attack epochs while VPG victim requires 10 epochs for being attacked successfully, with the strategy learned on a DQN proxy. As shown in Figure 4.9, with the strategy learned on a PPO proxy, DQN victim is successfully attacked within 2 attack epochs whereas VPG victim's policy is manipulated using 10 epochs. In summary, these observations indicate that an DBB-EPA strategy, which is learned on an on-policy DRL agent, can be successfully transferred to poison an off-policy DRL agent, and vice versa.

Furthermore, we notice that the attack efficiency varies for different victims, which is attributable to the characteristics of the victim's learning algorithm. Specifically,



(A) Transfered attack on a *PPO* victim

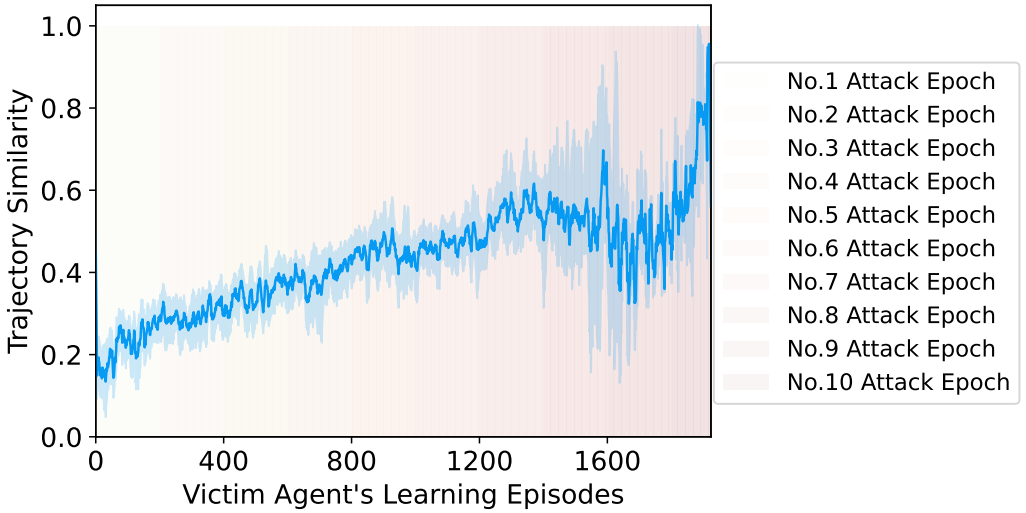


(B) Transfered attack on a *VPG* victim

FIGURE 4.8: Transferability evaluation of an attack strategy learned from a black-box *DQN* proxy agent in LunarLander.

the off-policy DQN algorithm learns its policy at each timestep using samples in a replay buffer. The on-policy PPO algorithm updates its policy with each sample of a trajectory at the end of the episode. The VPG algorithm, on the other hand, updates its policy based on one return of an entire trajectory at each episode. Due to the different frequency of policy updating, VPG's policy learning process is slower than that of DQN or PPO. This explains why VPG victim requires more attack epochs to obtain the target policy. In summary, the DBB-EPA strategy can be transferred to poison different DRL agents' policies, nevertheless, the attack efficiency depends on the characteristics of the victim's specific learning algorithm. This conclusion is consistent with the transferability discussion for tabular RL in

(A) Transfered attack on a *DQN* victim.

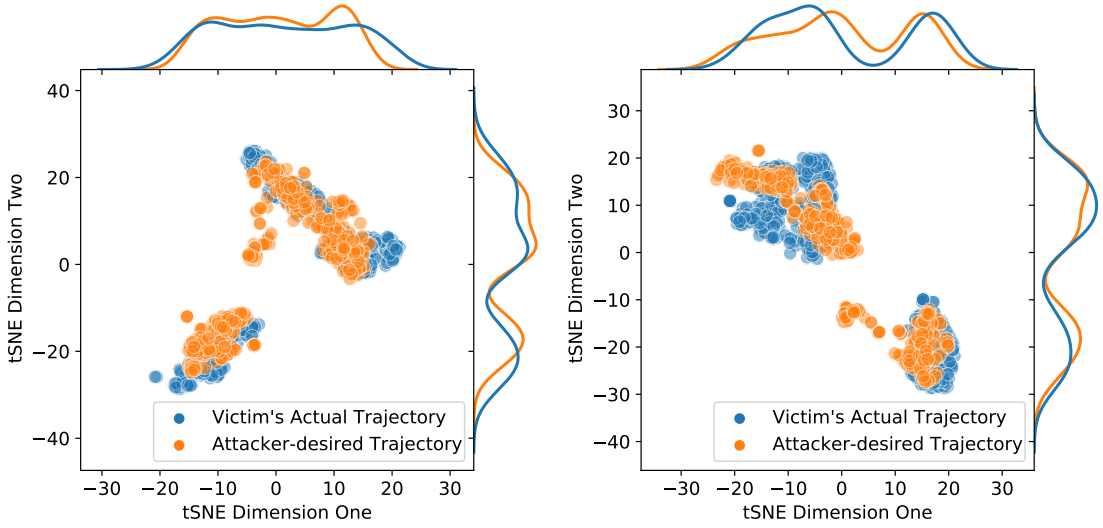


(B) Transfered attack on a *VPG* victim.

FIGURE 4.9: Transferability evaluation of an attack strategy learned from a black-box *PPO* proxy agent in LunarLander.

the discrete domain (as Section 3.6.3.1).

Visualization of Poisoned Policy. To examine the similarity between the victim's poisoned trajectory with the attacker-desired one, we use tSNE [98] to visualize their representations in the latent space. We collect 500 trajectories generated by the victim's poisoned policy and the attacker-desired policy, respectively. Individual trajectory is encoded as an embedding and visualized by tSNE. In Figure 4.10a and 4.10b, the victim's poisoned trajectories are clustered same as the attacker-desired ones, which has been shown by the overlapping distribution curves



(A) tSNE visualization for a DQN victim

(B) tSNE visualization for a PPO victim

FIGURE 4.10: Visualization of the victim's poisoned policies and the attacker-desired trajectories.

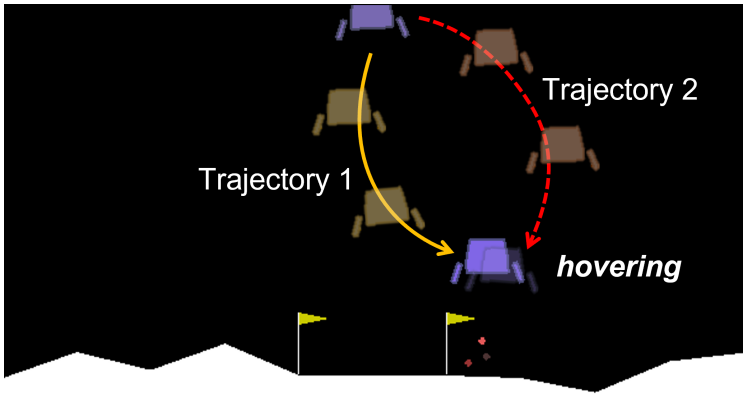


FIGURE 4.11: Illustration of attacker-desired trajectories.

along the two tSNE dimensions. Notice that two clusters of embeddings exists, which represent the two kinds of attacker-desired trajectories as shown in Figure 4.11, i.e., the spaceship experiences two kinds of trajectories and finally keeps hovering around the right flag more than 1000 time steps. In conclusion, both the DQN agent and the PPO agent have been misled to learn the attacker-desired policies in the poisoned training environment, and both experience the attacker-desired trajectories in the natural testing environment.

4.5 Summary

In this chapter, we propose an environment-poisoning attack against an RL agent at training time, with minimal prior knowledge of the RL system. Assuming only the ability to alter the environment hyper-parameters, our attack achieves minimal and adaptive environment poisoning, forcing a *black-box* RL agent to learn an attacker-desired policy in an *unknown* environment. Empirically, it achieves comparable performance to that of the white-box attack (i.e., TEPA) in the grid world, and succeeds in poisoning the DRL agent's policy in continuous environments. This study investigates a more realistic security threat posed by environment hyper-parameters, which can serve as a test-bed core to analyze vulnerabilities of RL to training-environment poisoning.

Chapter 5

Policy Resilience to Environment Poisoning Attack

5.1 Introduction

To guarantee the security associated with the learning of RL policies, defenses against training-time attacks have been developed from the standpoint of *robustness* which refers to the ability of an agent to maintain its functionality in the presence of perturbations [73] (illustrated as Figure 5.1). These robustness-based defenses [61, 73–80, 82] either theoretically or empirically guarantee the performance of learning policy under perturbations at training time. In spite of the fact of robustness is a crucial issue, it is merely an add-on concern when designing RL algorithms, which could increase design costs or compromise policy performance. Thus, there may be a lack of resources or priority in incorporating robustness into the design of RL systems. In addition, a successful robustness must be prepared to defend against all vulnerabilities and their associated attacks. However, the majority of vulnerabilities are unknown until they are exploited by an attacker, or known but unable to be prevented in advance. As a result, it is difficult and costly to achieve complete robustness, especially when attack approaches have been developed intensively. In light of these constraints, relying solely on robustness mechanisms is insufficient to protect the learning of RL policies from being poisoned. There is a need to shift the focus of security from robustness to *resilience* which

in this context refers to an RL agent's ability to recover its policy from malicious manipulations [73] (see Figure 5.1).

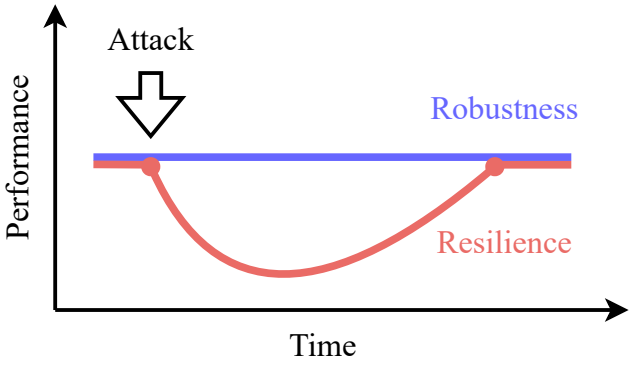


FIGURE 5.1: Illustration of robustness and resilience.

In this chapter, we attempt to address environment-poisoning attacks from the perspective of resilience, seeking an optimal deployment performance in the event of successful training-time attacks. Note that we specifically focus on the attacks that manipulate hyper-parameters of training environments, which has been described in Chapter 3 and Chapter 4. In these attacks, the dynamics of training environment are manipulated as a function of hyper-parameters, thereby the poisoned environment still shares the common environment structure (e.g., function) as the natural one. We therefore ask the following question: Can we design an efficient policy-resilience mechanism against such an environment-poisoning attack by exploiting the common structure knowledge of training environments?

This chapter provides a solution to this question. We propose a policy-resilience mechanism in which the poisoned RL agent exploits the environment-structure knowledge to quickly grasp the dynamics of the deployment environment and consequently recover its policy for optimal deployment performance. In order to facilitate the agent's access to environment-structure knowledge efficiently and effectively, we design the mechanism using a federated framework. Specifically, the proposed policy-resilience mechanism organizes independent and isolated RL systems in a federated manner. Within each of these RL systems, the environment dynamics share a common underlying structure and are represented as a parameterized function of hidden parameters (i.e., hyper-parameters), which can be formally defined as an instance of Hidden Parameter Markov Decision Process (HiP-MDP) [24]. As a result of these RL systems being organized in a federated manner, the common

environment-structure knowledge can be extracted in the center server and shared among isolated RL systems. For instance, autonomous vehicles could be organized by a transportation agency in a federated manner, as shown in Figure 5.2. As a center server, the transportation agency can collect environment data (e.g., road conditions) from a variety of vehicles without compromising data privacy, and extract useful environment knowledge to share with each vehicle to assist it driving in different environment instances. Similar examples include robots connected via a cloud server, buildings organized by a smart city center, and financial companies supervised by a central bank.

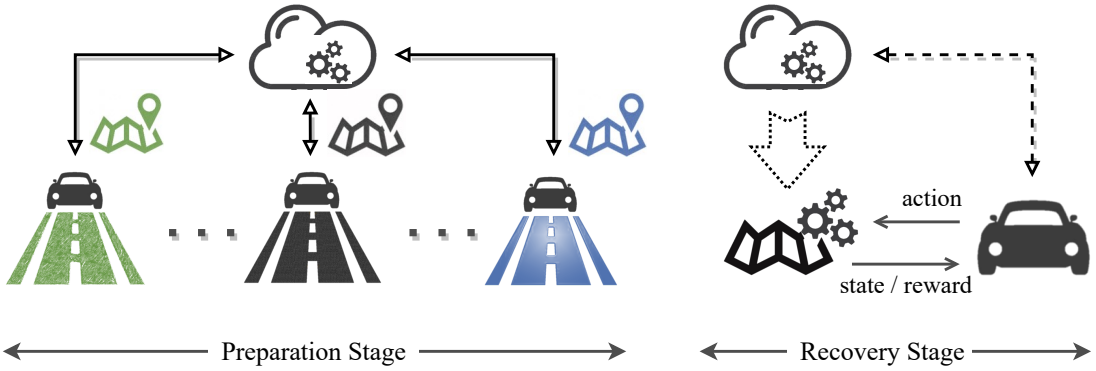


FIGURE 5.2: An example of policy-resilience mechanism in RL systems.

The procedure of our policy-resilience mechanism can be decomposed into three stages, which includes preparation, diagnosis and recovery. As shown in Figure 5.2, preparation is the first step which intend to extract the critical knowledge of environments within a set of RL systems. This knowledge is derived from federated RL systems with the help of a meta-learning approach. The diagnosis stage occurs prior to the deployment of the learned policy. During this stage, an RL agent fetches environment information from the federal server in order to efficiently understand the dynamics of a deployment environment that has been encountered. As shown in Figure 5.2, the poisoned policy is recovered in response to an accurate understanding of environment dynamics, resulting in an optimal behaviour in the deployment environment.

As a summary, this chapter makes the following contributions:

- We attempt to provide an RL agent with resilience ability that attempt to compensate for the imperfect robustness against training-time attacks, by allowing the agent to recover its policy from malicious manipulations prior to deployment.

- We propose a policy-resilience mechanism specifically for environment poisoning attacks. In this mechanism, independent RL systems are organized into federated systems, which allows each of them to exploit shared knowledge to facilitate the identification of deployment environments and the recovery of individual policies.
- The policy resilience mechanism can be described as a three-step pipeline that includes preparation, diagnosis and recovery. Meta-learning approach is incorporated with federated manner in the stage preparation and diagnosis to ensure an efficient fusion of environment knowledge and an accurate interpretation of deployment dynamics.
- An empirical evaluation of the proposed policy-resilience mechanism shows that, in natural deployment environments, a poisoned policy can be effectively recovered for optimal performance.

5.2 Preliminaries

Hidden-Parameter MDP. A Hidden Parameter MDP (HiP-MDP) [99] represents a family of MDPs in which hidden parameters $e \in \mathbb{R}^n$ are used to parameterize the transition dynamics. As examples of hidden parameters, we can mention gravity, friction on a surface or the strength of a robot actuator. These parameters are not part of the observation space but play a significant role in the response of the environment to the actions of agents [22]. Note that the hidden parameter shares the same definition of hyper-parameters in TEPA and DBB-EPA, thereby we use the expression hyper-parameter uniformly in the following. Formally, a HiP-MDP can be defined as a tuple $\langle S, A, R, F_e, \gamma \rangle$, in which S is the set of states, A is the set of actions, and R is a reward function. $F(s'|s, a, e_i)$ is a transition function for each task instance i parameterized by the hyper-parameter e_i. Here, the parameter e_i is drawn from a prior distribution $e_i \sim p(e)$. The HiP-MDP framework assumes that variations in the dynamic of true tasks can be fully described by a finite-dimensional array of hidden parameters [99].

Federated Learning. The main idea of Federated Learning (FL) [100] is to utilize distributed data sets to generate machine learning models with protection

of data privacy and security. FL succeeds to solve the dilemma that the data is distributed at isolated islands but is forbidden to be collected/fused for processing model [101]. Recently, FL has addressed data isolation and privacy issues in RL-based applications, such as smart energy management [5], autonomous driving in Internet of Vehicles (IoV) [102] and optimal control of internet-of-things devices (IoT) [103].

Meta Learning. The meta-learning method involves learning a model from an array of tasks in order to make it capable of solving new tasks with only a limited number of training examples [104]. Using meta-learning, a parameterized algorithm (i.e., meta-learner) is learned by performing a meta-training process, and it can be used to fast train a specific model in each task. Specifically, each task consists of two distinct data sets: a support data set and a query data set. A task-specific model is trained on the support set and then tested on the query set. Using these test results, the parameterized algorithm is updated. It is important to note that the parameterized algorithm is capable of learning how to adapt to new tasks more quickly, i.e., "learning to fine-tune".

5.3 Problem Statement

In this section, we first describe the settings for both the attacker and victim RL agents. Next, we provide a formal description of the federated design of the policy-resilience framework, as well as an introduction to the policy-resilience procedure.

5.3.1 Attacker and Victim

Our policy-resilience mechanism is proposed to address the negative effects caused by an environment-poisoning attack. The attacker and the victim are described as follows.

Attacker: To poison an RL agent's policy, the attacker manipulates the agent's environment hyper-parameters when the agent learns its policy. It aims to force the RL agent to obtain the attacker-desired policy with minimal changes to the victim's environment. Here, if the attacker is assumed to have prior knowledge of

the RL system, it can learn an optimal attack strategy following TEPA proposed in Chapter 3; otherwise, the attack strategy can be learned by DBB-EPA proposed in Chapter 4.

Victim: The victim RL agent aims to learn an optimal policy that maximizes cumulative discounted rewards. We assume that the agent is oblivious to the attack and continues to operate normally throughout the sequence of environment modifications. Policy-resilience mechanisms are applied after the agent has learned its policy and before the agent officially deploys the policy. More details about the policy-resilience implementation will be presented in Section 5.3.3.

5.3.2 Policy-Resilience Framework

Our policy-resilience framework is designed in a federated manner, which organizes multiple independent and isolated RL systems. The environments of these RL systems are required to be instances of the same HiP-MDP. As described in Section 5.2, environments that have a common structure but differ in hyper-parameters can be also considered as instances in a HiP-MDP. Under TEPA or DBB-EPA, the agent's environment hyper-parameters are altered while the environment structure remains unchanged. Thus, the poisoned environment at each attack epoch can be regarded as an instance of HiP-MDP.

As shown in Figure 5.4, the policy-resilience framework consists of one central server and a set of client RL systems, in which there are learnable dynamics models in the server and each RL system. We formalize the framework as $\langle \mathcal{S}, \{\mathcal{C}_i\}_{i=1}^{K} \rangle$, where

- $\mathcal{S}$ represents a dynamics model in the reliable server, which is a function parameterized by $\theta_{\mathcal{S}}$.
- $\{\mathcal{C}_i\}_{i=1}^{K}$ represents a set of client RL systems governed by the server, where K is the number of systems.

Here, each client system $\mathcal{C}_i$ includes an RL agent, a Markovian environment, a data buffer and a dynamics model, which is described as a tuple $\mathcal{C}_i = \langle \mathcal{R}_i, \mathcal{M}_i, \mathcal{T}_i, \mathcal{D}_i \rangle$, where

- $\mathcal{R}_i$ is an RL agent that seeks to learn a policy π to maximize cumulative rewards using either model-free or model-based learning algorithms.
- $\mathcal{M}_i$ denotes a Markovian environment $\langle S, A, F_{e_i}, R, \gamma \rangle$. Here, S and A are sets of the RL agent's states and actions, respectively. F_{e_i} represents the environment dynamics parameterized by the environment hyper-parameters e_i. R is the reward function and γ is the discount value. It is important to note that $\{\mathcal{M}_i\}_{i=1}^{K}$ are in the same HiP-MDP with $e_i \sim p(e)$ affecting the dynamics function F_{e_i}.
- $\mathcal{T}_i$ is a buffer that stores the RL agent's transitions $\langle s, a, s', r \rangle$ used for learning a local dynamics model.
- $\mathcal{D}_i$ represents a local dynamics model that shares the same structure as the server dynamics model $\mathcal{S}$.

In the design of federated framework, each RL agent is forbidden to directly interacts with other agents' environment or access raw transition data in other systems. Based on the trustworthy server, clients share information (i.e., loss value or gradients) with the server and trust the knowledge (i.e., parameters of the dynamics model) retrieved from the server. Additionally, the poisoning in each RL agent can be customized by the attacker.

In this context, we assume that the security of the federated framework is assured, so that we can concentrate on studying the security threats occurring in the isolated single RL system.

5.3.3 Policy-Resilience Procedure

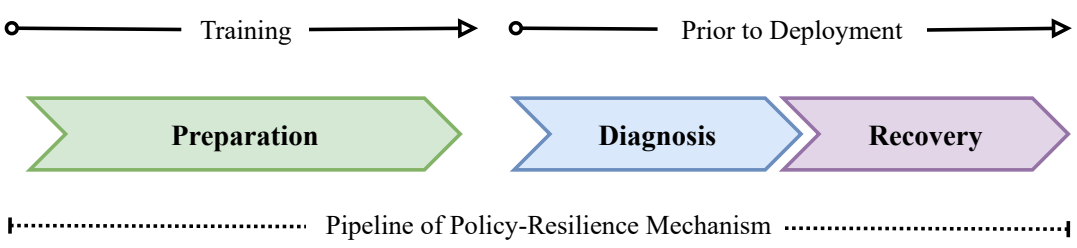


FIGURE 5.3: Pipeline of policy-resilience mechanism.

Based on the design of federated framework, our proposed policy-resilience procedure involves three stages: preparation, diagnosis, and recovery, as illustrated in Figure 5.3.

Preparation occurs during the training process of RL policies, which attempts to extract critical knowledge about the environment structure. At this stage, we utilize a meta-learning mechanism to learn the server dynamics model $\mathcal{S}$ based on environment information from all the federated RL systems. With the use of the meta-learning approach [104], the server dynamics model can be learned as a parameterized function, which can be easily adapted to environments that share a common structure but differ in hyper-parameters. A high-quality dynamics model can facilitate the fast modelling of unseen deployment environments and the effective recovery of policies at later stages.

Diagnosis is the second stage, which is carried out prior to the deployment of the learned policy. To accurately capture the dynamics model of a deployment environment, the agent will retrieve environment knowledge from the server and personalize the knowledge to its deployment environment dynamics. Note that diagnosis phase is time- and resource-sensitive, during which deployment environment dynamics should be understood with the least amount of agent-environment interaction as possible.

Recovery is the final stage of the policy-resilience mechanism, which aims to reduce the negative effects of the training-time environment poisoning and restore policy deployment performance to the full extent of its ability. The recovery of a policy is accomplished using the trajectories imagined from the deployment dynamics model, so the quality of diagnosis is critical.

5.4 Methodology: Policy-resilience Mechanism

In this section, we present the method of our policy-resilience mechanism in accordance with the procedure described above. First, we describe mathematically how the common environment structure is acquired in the federated framework using a meta-learning mechanism (i.e., the preparation stage). After that, we describe how the poisoned policy can be recovered in response to an accurate understanding of the agent's deployment environment (i.e., the diagnosis and recovery stages).

5.4.1 Preparation

The intention of preparation is to learn a server dynamics model, which captures the underlying structure of training environments, by employing a meta-learning methodology. Figure 5.4 illustrates the learning of the server dynamics function based on environment information collected from all the client RL systems. Specifically, the process whereby the server transmits information to client systems and then receives messages from them is regarded as a *round*. As shown in Figure 5.4, each round consists of four steps:

- (1) the transfer of dynamics-model parameters (from the server to clients);
- (2) the calculation of loss value (in each client system);
- (3) the collection of loss values (from clients to the server);
- (4) the updating of dynamics-model parameters (in the server).

As we follow these four steps at each round, we learn the server dynamics model through a combination of federated learning and meta-learning approaches [104].

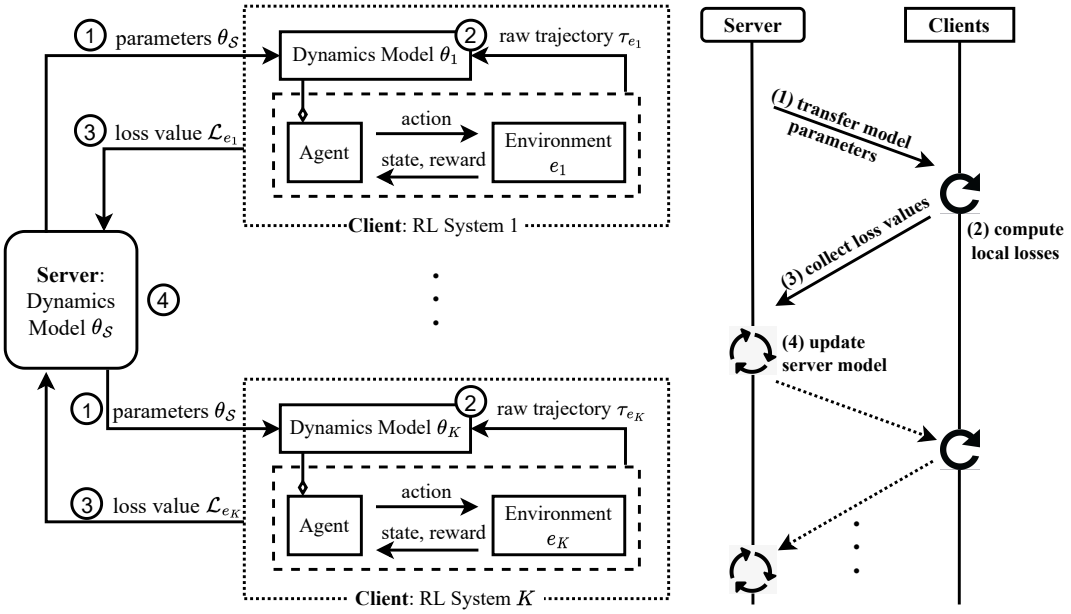


FIGURE 5.4: Illustration of preparation stage in policy-resilience mechanism.

At the 1^{st} step of each round, local dynamics models $\{\mathcal{D}_i\}_{i=1}^{K}$ should be initialized with the parameters $\theta_{\mathcal{S}}$ which is retrieved from the server dynamics model $\mathcal{S}$. At the

2^{nd} step, after the initialization of each local dynamics model $\mathcal{D}_{\theta_i=\hat{\theta}}$, loss values are locally calculated in a meta-learning manner. Specifically, in each client RL system where environment is parameterized by hyper-parameters e_i, transitions $\langle s, a, s' \rangle$ are collected during the learning of RL agent's policy. These transitions are saved in the local buffer $\mathcal{T}_i$, which separated into a support set $\mathcal{T}_{e_i}^{spt} = \{\langle s_m, a_m, s'_m \rangle\}_{m=1}^{M}$ and a query set $\mathcal{T}_{e_i}^{qry} = \{\langle \bar{s}_n, \bar{a}_n, \bar{s}'_n \rangle\}_{n=1}^{N}$. The local dynamics model $\mathcal{D}_{\theta_i=\theta_\mathcal{S}}$ is trained on the support set $\mathcal{T}_{e_i}^{spt}$ using a gradient-descent learning algorithm, which generates updated parameters $\theta_i^{'}$. The updated local dynamics model $\mathcal{D}_{\theta_i^{'}}$ is then evaluated on the query set $\mathcal{T}_{e_i}^{qry}$ and thereby a loss value $\mathcal{L}(\mathcal{T}_{e_i}^{qry}, \mathcal{D}_{\theta_i^{'}})$ is computed. The local loss value reflects the learning ability of $\mathcal{D}_{\theta_i}$ which is parameterized by $\theta_\mathcal{S}$.

At the 3^{rd} and the 4^{th} step, the server aggregates all the loss values from federated RL systems, computes the gradient and then updates the server dynamics model. Specifically, loss values $\left\{\mathcal{L}(\,\cdot\,, \theta_i^{'})\right\}_{i=1}^{K}$, which are obtained in all the client RL systems, are collected and transferred to the server. Based on these loss values, the federal dynamics model $\mathcal{S}(\theta_\mathcal{S})$ are optimized to minimize the aggregated loss as following objective:

$$\begin{aligned} &\min_{\theta_\mathcal{S}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(\mathcal{T}_{e_i}^{qry}, \mathcal{D}_{\theta_i^{'}}) \\ &\qquad \text{s.t.} \\ &\theta_i^{'} = \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}(\mathcal{T}_{e_i}^{spt}, \mathcal{D}_{\theta_i=\theta_\mathcal{S}}), \end{aligned} \tag{5.1}$$

where α is the learning rate. When the parameters $\theta_\mathcal{S}$ are optimized, the server dynamics model $\mathcal{S}$ is a function parameterized by $\theta_\mathcal{S}^*$. It is capable of efficiently modelling an environment $e \sim p(e)$ using a small number of transition data.

In addition, we provide a specific definition of the loss function $\mathcal{L}(\mathcal{T}_e, \mathcal{D}_\theta)$ depending on the type of environment state domain, i.e., continuous state domains and discrete state domains. For example, in continuous state domains, we measure the loss value via Mean Square Error (MSE) as

$$\mathcal{L}(\mathcal{T}_e, \mathcal{D}_\theta) = \sum_{j=0}^{X} \left\| \mathcal{D}_\theta(s_j, a_j) - s_{j+1} \right\|_2^2, \tag{5.2}$$

where X is the number of transitions in $\mathcal{T}_e$. Similarly, the loss value is measured by cross entropy in discrete state domains, which is denoted as

$$\mathcal{L}(\mathcal{T}_e, \mathcal{D}_\theta) = -\mathbb{E}_{\langle s,a,s' \rangle \sim \mathcal{T}_e} \log \mathcal{D}_\theta(s'|s,a) \tag{5.3}$$

In summary, the learning of the federal dynamics model is finalized as Algorithm 7. We consider the learned dynamics model on the server as being representative of the common underlying structure of the training environments in federated RL systems.

Algorithm 7 Preparation: Learning of the Server Dynamics Model $\mathcal{S}$

Require: Client RL systems $\{\mathcal{C}_i\}_{i=1}^K$
Initialize the global dynamics model $\mathcal{S}(\theta_\mathcal{S})$
for n_round = 1,2,... **do**
  for each client $\mathcal{C}_i$ in parallel **do**
    Initialize $\mathcal{D}_i$ with $\theta_\mathcal{S}$ retrieved from $\mathcal{S}$
    if n_round mod n_interval = 0 **then**
      Collect X transitions $\langle s, a, s' \rangle$
      $\mathcal{T}_{e_i} \leftarrow \mathcal{T}_{e_i} \cup \{\langle s_j, a_j, s'_j \rangle\}_{j=1}^X$
    end if
    Sample $\left\{\mathcal{T}_{e_i}^{spt}, \mathcal{T}_{e_i}^{qry}\right\}$ from $\mathcal{T}_{e_i}$
    Update local parameters $\theta_i^{'} \leftarrow \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}(\mathcal{T}_{e_i}^{spt}, \mathcal{D}_{\theta_i = \theta_\mathcal{S}})$
    Compute loss values $\mathcal{L}(\mathcal{T}_{e_i}^{qry}, \mathcal{D}_{\theta_i^{'}})$ following Eq.[5.2] or Eq.[5.3]
    Send the loss value $\mathcal{L}(\cdot, \theta_i^{'})$ to server $\mathcal{S}$
  end for
  In server, aggregate losses $\mathcal{L}_\mathcal{S} \leftarrow \frac{1}{K} \sum_{i=i}^K \mathcal{L}(\cdot, \theta_i^{'})$
    update parameters $\theta_\mathcal{S}^{'} \leftarrow \theta_\mathcal{S} - \beta \nabla_{\theta_\mathcal{S}} \mathcal{L}_\mathcal{S}$
end for
Return parameters $\theta_\mathcal{S}^*$ of global dynamics model $\mathcal{S}$

5.4.2 Diagnosis

In the diagnosis stage, the agent attempts to gain an understanding of the dynamics of its deployment environment by relying on the information obtained from the server, as well as the transitions collected locally. The diagnosis stage consists of three steps:

- the retrieval the knowledge (i.e., parameters) from the server dynamics model;

- the collection of trajectories in the deployment environment;
- the understanding of the deployment environment (i.e., dynamics model).

Imagine, for example, a RL agent who has learned its policies in a poisoned training environment. The agent should understand the dynamics model of the deployment environment, prior to executing the policy in a natural deployment environment containing hyper-parameters $\hat{e} \sim p(e)$, in order to guide its recovery. Specifically, the agent first initializes its local dynamics model using the parameters $\theta^*_{\mathcal{S}}$ which is retrieved from the server dynamics model $\mathcal{S}$. After the initialization, the parameters of this local dynamics model $\mathcal{D}_{\theta_{\hat{e}}=\theta^*_{\mathcal{S}}}$ are fine-tuned based on a small set of transitions $\mathcal{T}_{\hat{e}} = \{\langle s_n, a_n, s_{n+1}\rangle\}_{n=1}^{N}$ sampled in the deployment environment, which is denoted as

$$\theta'_{\hat{e}} = \theta_{\hat{e}} - \alpha \nabla_{\theta_{\hat{e}}} \mathcal{L}(\mathcal{T}_{\hat{e}}, \mathcal{D}_{\theta_{\hat{e}}=\theta^*_{\mathcal{S}}}),$$

where α is the learning rate and $\mathcal{L}$ is defined as Equation 5.3 or Equation 5.2.

Since the diagnosis stage is time-sensitive and resource-sensitive, we expect that the agent can quickly understand the dynamics model of its deployment environment, using a small number of transition data. The diagnosis performance relies on the knowledge quality retrieved from the server, i.e., the learning ability of the server dynamics $\mathcal{S}$ parameterized by $\theta^*_{\mathcal{S}}$.

5.4.3 Recovery

The goal of the recovery stage is to restore the deployment performance of poisoned policies to their full potential. The recovery of policy depends on imagined trajectories, therefore an understanding of the dynamics of the deployment environment (i.e., diagnosis) is crucial to finding an optimal policy. In this section, we present possible recovery solutions from the perspective of a model-free RL agent and a model-based RL agent, respectively.

To begin with, we describe the imagined trajectories generated by the dynamics model $\mathcal{D}_{\theta'_{\hat{e}}}$ that has been tailored to the deployment environment during the diagnosis phase. In a deterministic environment, an agent utilizes the dynamics model to predict the future state s' based on the current state s and the chosen action a, denoted as $s' \sim \mathcal{D}_{\theta'_{\hat{e}}}(s, a)$. In a stochastic environment, the dynamics model may

be designed as an uncertainty-aware neural network [105, 106], which consists of ensembles of independently trained models rather than a single model. The agent can predict the future states with consideration of uncertainty which is estimate via the mean and deviation of multiple stochastic forward passes of the models. The state prediction is denoted as $s^{'} \sim \mathcal{N}\left(\mathbb{E}\left[\mathcal{D}_{\theta'_{\hat{e}}}(s,a)\right], \sqrt{\text{Var}\left[\mathcal{D}_{\theta'_{\hat{e}}}(s,a)\right]}\right)$ where $\mathcal{N}$ represents Gaussian distribution. Therefore, in both deterministic and stochastic environment, the agent is able to recursively predict future states $\{s_{t+1}, \ldots, s_{t+H}\}$ via an action sequence $\{a_t, \ldots, a_{t+H-1}\}$ given an initial state s_0, generating imagined trajectories.

Model-Free RL Agent. For a model-free RL agent that attempts recover its policy using imagined trajectory, we adopt a model-based value expansion (MVE) [107, 108] as the approach The MVE approach incorporates the dynamics model $\mathcal{D}_{\theta'_{\hat{e}}}$ into value estimation by replacing the standard Q-learning target with an improved one V_h^{MVE}. V_h^{MVE} is computed by rolling the dynamics model $\mathcal{D}_{\theta'_{\hat{e}}}$ out for h steps, which limits imagination to a fixed depth h so as to prevent accumulative errors due to inaccuracies in the dynamics model. Specifically, the value estimation $V_h^{MVE}(r, s')$ is composed of a short-term value estimate derived from the unrolling of the dynamics model $\mathcal{D}_{\theta'_{\hat{e}}}$, and a long-term value estimate derived from the learned values $Q^{\pi}_{\theta^-}$, denoted as

$$V_h^{MVE}(r, s') = r + \left(\sum_{i=1}^{h} T^i \gamma^i R(s'_{i-1}, a'_{i-1}, s'_i)\right) + T^{h+1}\gamma^{h+1} Q^{\pi}_{\theta^-}(s'_h, a'_h), \tag{5.4}$$

where $s_i \sim \mathcal{D}_{\theta'_{\hat{e}}}(s_{i-1}, a_{i-1})$ and $Q^{\pi}_{\theta^-}$ is an approximated action-value function. T is the termination of the trajectory and R represents a reward function, which are assumed to be known in this context. Note that T and R also can be learned based on trajectories.

Model-Based RL Agent. The learning algorithm adopted by a model-based RL agent is Model Predictive Control (MPC) with cross-entropy method (CEM) [109]. MPC is an online learning approach which can re-plan an action based on updated state information, so that it can prevent accumulative errors caused by the dynamics model $\mathcal{D}_{\theta'_{\hat{e}}}$. Specifically, at time step t, the MPC approach iteratively samples action $a_{t:t+H-1}$ from multivariate normal distributions $\mathcal{N}(a_t|\mu_t, \sigma_t)$ which

is adjusted based on the best sampled actions. Here, μ_t and σ_t are the mean and the variance of top 10% actions. In MPC, the agent implements the first action from the optimal action sequence $a_{t:t+H-1}$ and then optimally re-plans an action sequence at time step $t+1$. Finally, the agent aims to find an action sequence which optimizes

$$\sum_{t}^{t+H-1} \mathbb{E}_{s_{t+1}\sim\mathcal{D}_{\theta^*_{\hat{e}}}(s_t,a_t)}\big[r(s_t,a_t,s_{t+1})\big] \tag{5.5}$$

using predicted state s_{t+1} from the local dynamics model $\mathcal{D}_{\theta^*_{\hat{e}}}$.

Based on the solutions outlined above, either a model-free or a model-based agent can recover its poisoned policy using imagined trajectories. Algorithm 8 summarizes the operations performed during the diagnosis and recovery stages.

Algorithm 8 Diagnosis and Recovery

Require: An optimized server dynamics model $\mathcal{S}(\theta^*_{\mathcal{S}})$
Require: An RL system with dynamics model $\mathcal{D}_{\theta_{\hat{e}}}$ and poisoned policy $\pi_{\hat{e}}$
1: Initialize $\mathcal{D}_{\theta_{\hat{e}}}$ with $\theta^*_{\mathcal{S}}$ retrieved from $\mathcal{S}$
2: Collect n-episode transitions into data set $\mathcal{T}_{\hat{e}}$
3: Update parameters of $\mathcal{D}_{\theta_{\hat{e}}}$ as $\theta'_{\hat{e}} = \theta_{\hat{e}} - \alpha\nabla_{\theta_{\hat{e}}}\mathcal{L}(\mathcal{T}_{\hat{e}}, \mathcal{D}_{\theta_{\hat{e}}=\theta^*_{\mathcal{S}}})$
4: Recover $\pi_{\hat{e}}$ following Eq. 5.4 or 5.5, using trajectories imagined via $\mathcal{D}_{\theta'_{\hat{e}}}$

5.5 Experiment

We empirically evaluate our proposed policy-resilience mechanism by verifying the following questions: (1) Can our approach recover the poisoned RL policy from malicious manipulation caused by corrupted training environments? (2) Does the sharing of environment knowledge enable a poisoned agent to grasp the dynamics of deployment environments more efficiently for policy recovery? (3) Does the meta-learning mechanism provide an effective means of extracting critical knowledge about environment dynamics within a federated framework? In this section, we provide empirical answers to these questions in both discrete and continuous state domains.

5.5.1 Discrete State Domains

This part evaluates the design and performance of a policy-resilience mechanism in discrete state domains.

5.5.1.1 Experiment Settings

Our proposed policy-resilience mechanism is targeting the environment-poisoning attacks at training time, particularly those attacks which poison the environment dynamics by perturbing the parameters of the physical environment (i.e., the hyper-parameters). The attacked RL agent is learning a navigation task in a 3D grid worlds with size 5×5 as shown in Figure 3.4. Details of the attack approach and the experiment environment can refer to Chapter 3.

Baselines: Due to the limited number of studies that examine policy resilience against environmental poisoning, there is no appropriate baseline that can be directly referenced. Thus, we design two types of baselines in accordance with our experimental objectives.

- Policy-Resilience Mechanism without Federated Framework: To evaluate whether the federated structure is necessary for extracting and sharing environment knowledge, we design a baseline policy-resilience mechanism which does not rely on server-based environment knowledge. Thus, both during the preparation and diagnosis stage, the agent learns the environment dynamics model locally without shared knowledge, using local Stochastic Gradient Descent (SGD) updates. This baseline is termed as *No Federated Framework*.

- Policy-Resilience Mechanism without Meta-learning Mechanism: To evaluate the effect of a meta-learning mechanism on the quality of environment-knowledge extraction and fine-tuning, we design a baseline policy-resilience procedure where the federal dynamics model is learned using Federated Averaging Learning [100]. It means that during the preparation process, the federal dynamics model is optimized by averaging local SGD updates. This baseline is termed as *No Meta-Learning Mechanism*.

5.5.1.2 Experiment Results

For these experiments, the policy-resilience framework includes 10 client RL systems that are all assumed to belong to the same HiP-MDP. We initialize the hyperparameters of these RL environments at the same values in order to simplify the discussion. All of them are involved in the preparation process, and some of them may be poisoned during training.

Performance of Policy Resilience. We aim to evaluate the recovery performance of one RL agent's poisoned policy when different proportions of poisoned agents are set at preparation stage. Figure 5.5 shows that even though the majority of RL agents have been attacked during training (e.g., the poison proportion is equal to 80%), their poisoned policies can be effectively recovered to a good performance. In Figure 5.5, we can see that the policy performance (i.e., mean returns) increases to an optimal level within two episodes. This means that the attacker successfully understands the deployment environment and recovers the poisoned policy, only based on two-episode transition data which is of limited amount (i.e., not more than 28). Such a time-efficient and resource-efficient recovery is due to the knowledge sharing of the environment. Specifically, the server dynamics model, which is learned by extracting environment information from a set of client RL agents, provides beneficial knowledge to the agent. With this knowledge, the agent can quickly identify key details of the deployment environment for restoring policies using imagined trajectories. In addition, when all client RL agents are poisoned during training (i.e., poison proportion is equal to 100%), the policy-resilience performance is somewhat reduced, but it is still effective. The analysis of this performance will be presented in the upcoming discussion *effect of meta-learning design.*

Effect of Federated Design. We are investigating the effects of the shared knowledge which is achieved by the design of a federated framework. In this experiment, only one RL system has been attacked at the preparation stage and it transfers the information of the corrupted environment to the server. Figure 5.6 shows the performance comparison between our policy-resilience mechanism and the baseline *No Federated Framework*. As shown, our approach succeeds in recovering the performance of a poisoned policy based on a dynamics model that is

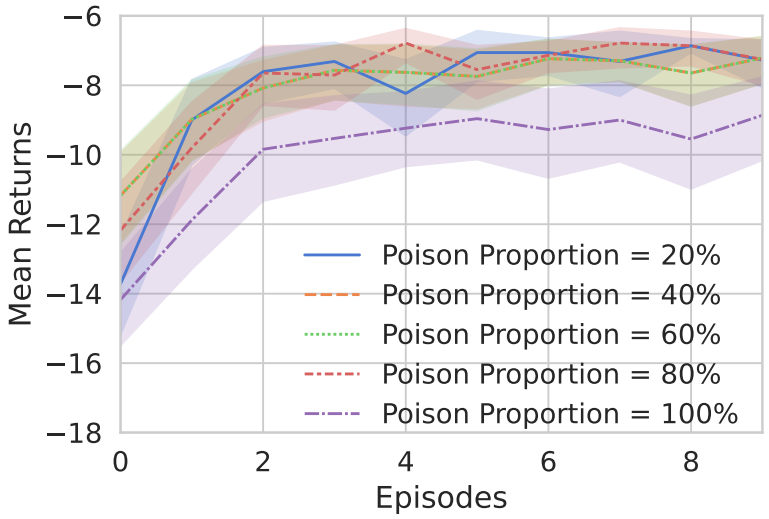


FIGURE 5.5: Policy-resilience performance when various proportions of RL systems are poisoned in the preparation stage.

initialized by the shared knowledge (i.e., parameters of server dynamics model) and fine-tuned using the local two-episode trajectory data. In contrast, the baseline *No Federated Framework* fails in recovering the policy when the agent is trying to improve its local dynamics model without the shared environment knowledge. These results demonstrate that shared knowledge of the deployment environment allows the agent to quickly and accurately understand the deployment environment, enabling imagined trajectories to be generated for policy recovery. The knowledge extraction and sharing require federated frameworks, so federated frameworks are essential and effective design elements.

Additionally, we would like to discuss the effects of federated design on an agent that is capable of meta-learning its dynamics model during policy training. In this case, the training environment has been poisoned by an attacker, as a result of which the agent has never experienced a natural environment when learning its policy. For the purpose of recovering its poisoned policy, the agent models the deployment environment only using its meta-learning dynamics model without shared environment knowledge. In this scenario, two cases should be considered in order to analyze recovery performance. Firstly, if the attacker poisons the agent's environment many times, the agent will encounter a variety of environments (i.e., HiP-MDP task instances) when it meta-learns the dynamics model. No one of these environments, however, is identical to the condition that all agents within the federated framework are poisoned (i.e., the poison proportion is equal to 100%). In

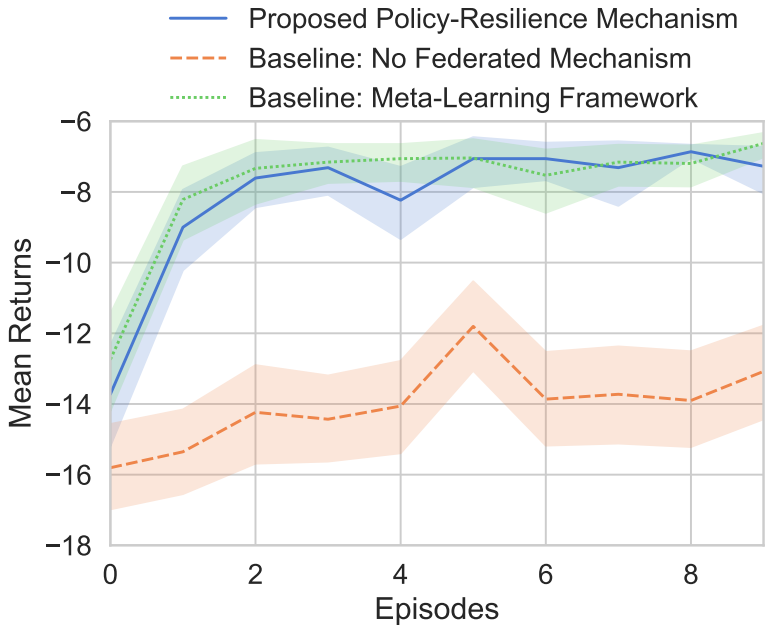


FIGURE 5.6: Comparison of policy-resilience performance.

this case, the policy-resilience performance has reduced, but it is still effective, as shown in Figure 5.6. Alternatively, if the attacker poisons the agent's training environment only once, the agent can only learn its dynamics model from one corrupted environment. While the agent learns the dynamics model through meta-learning, the model is not equipped with a learning ability as it has never been exposed to another task instance. Due to this limitation, the agent is unable to quickly understand its deployment and, therefore, cannot effectively recover its policy by using imagined paths. Considering these two cases, we find that a meta-learning agent's policy-resilience depends on the attacker's implementations, which are varying and outside of our control. Therefore, even if the agent has the capability to meta-learn the dynamics model, the federated design of policy recovery is necessary to ensure that policy recovery is independent of attack implementation.

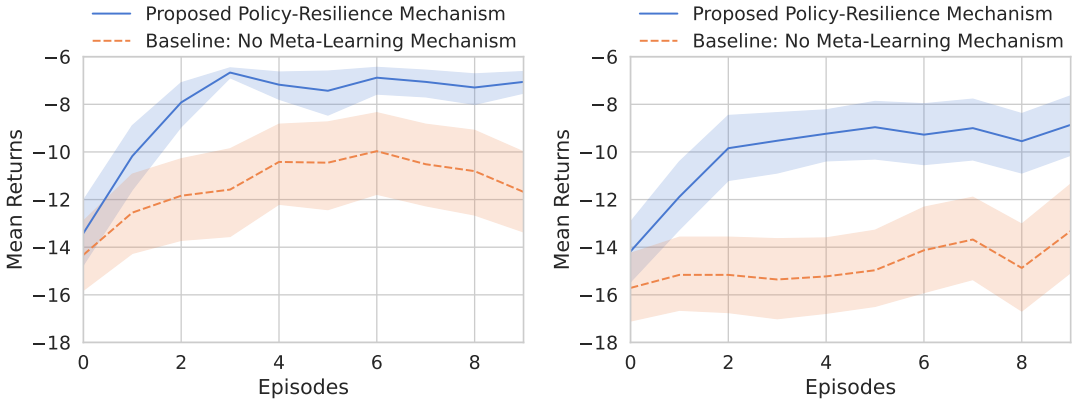
Actually, we have designed our policy-resilience mechanism in such a way that an agent in our RL will not need to invest additional resources or add-on capabilities for achieving resilience. Any agent may be provided with information on the environment and allowed to recover its poisoned policies accordingly, regardless of whether it participated in the preparation process. Thus, in the context of a single RL system, such a policy-resilience mechanism is resource-efficient.

Effect of Meta-Learning Design. In this experiment, we aim to evaluate the impact of the meta-learning approach on the quality of the federal dynamics model in the server.

First, Figure 5.6 shows that both the baseline *No Meta-Learning Mechanism* and our policy-resilience mechanism achieve good performances in recovering the policy. It means that, when only one RL system has been attacked at the preparation stage, the both approaches lead to effective knowledge extraction (i.e., the learning of a server dynamics model). Then, we observe that, as the proportion of poisoned client RL systems increases, the effect of meta-learning approach becomes more apparent. For example, as shown in Figure 5.7a, when the proportion of poisoned clients reaches 50%, our policy-resilience mechanism still succeeds in recovering the agent's poisoned policy, whereas the performance of baseline *No Meta-Learning Mechanism* is somewhat decreased. Here, this decreased performance can be attributed to the fact that the information collected from those poisoned RL systems has negatively impacted the learning of the server dynamics model. In contrast, the meta-learning approach is not a simple combination of the federated dynamics model. It considers these poisoned RL systems as task instances and learns to how to efficiently learn the dynamics model of each task instance (i.e., namely "learn to learn"). Based on the meta-learning process (see Eq. 5.1), the server dynamics model is a parameterized function which is optimized to grasp the dynamics feature from a small set of training data.

Furthermore, Figure 5.7b shows the policy-resilience performances when all client RL systems are poisoned at the preparation stage. Our policy-resilience mechanism is still effective in recovering the poisoned policy despite the slight reduction in efficiency and performance, while the baseline totally fails. This failure of the baseline indicates that the server dynamics model, which is learned using Federated Averaging Learning [100], cannot accurately model the natural deployment because the dynamics model is completely learned based on information generated by corrupted environments. In contrast, in our policy-resilience mechanism, the slight performance reduction can be attributed to the fact that the deployment environment is not included in the task sets at the preparation stage. In spite of this, the server dynamics model has been programmed with a learning ability, which enables it to model the dynamics of a task instance even if the task has not been

displayed during preparation. In conclusion, the inclusion of the meta learning in our policy-resilience mechanism is crucial.



(A) poisoning proportion = 50% (B) poisoning proportion = 100%

FIGURE 5.7: Comparison of policy resilience under various proportions of poisoned clients.

5.5.2 Continuous State Domains

In this part, we evaluate our policy-resilience mechanism on both model-free and model-based RL agents in continuous state domains.

5.5.2.1 Experiment Settings

Environments. The environment-poisoning attack, DBB-EPA, targets an RL agent performing a Cartpole task as depicted in Figure 5.8. It is the agent's goal to balance the cartpole by exerting force in either the left or right directions on it. When a force is applied, the angle and velocity of the pole are affected by the length of the pole. As a result, the pole length can be viewed as the environment hyper-parameter which could be maliciously altered by an attacker.

Baselines. In this experiment, we design two baselines to evaluate our proposed policy-resilience mechanism.

- Direct Policy Implementation without Resilience: To evaluate the performance of our policy-resilience mechanism, we use the policy learned in the

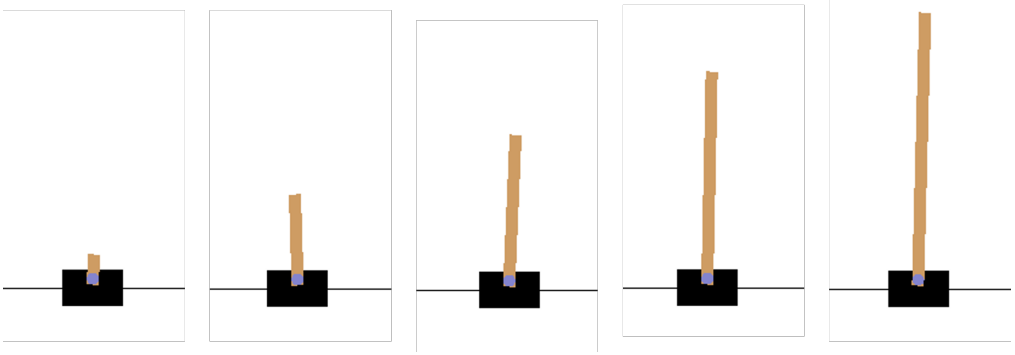


FIGURE 5.8: Illustration of Cartpole environment.

poisoned training environments as the baseline. In other words, the poisoned policy will be implemented directly in the natural deployment environment. Such a baseline is termed as *Baseline: no policy resilience*.

- Policy Resilience without Knowledge Sharing: To evaluate the design of federated framework associated with a meta-learning mechanism, we design a baseline policy-resilience procedure in which policy is recovered on the basis of its local dynamics model without the shared knowledge. We refer to this baseline as *Baseline: isolated policy resilience*.

5.5.2.2 Experiment Results

In light of the fact that an agent could adopt either model-free or model-based RL learning algorithms, we evaluate our policy-resilience mechanism on these two types of RL agents, respectively.

Model-free RL Agents. This experiment involves 10 client RL systems within the policy-resilience framework, all of whose environments have been manipulated by the attacker during preparation. We simplify the attack implementation by manipulating the pole length only once at the beginning of the agent's learning process. Thus, the RL agent learns its policy in the Cartpole environment where the pole length is manipulated as 0.5, but will use the learned policy to control a car in which the pole length is $0.1 \sim 0.9$ (see Figure 5.8). The agent learns the policy using model-free RL algorithms (e.g., DDPG) and recover it using MVE (see Chapter 5.4.3) based on imagined trajectories generated by the learned dynamics model.

Figure 5.9 illustrates the final performance of policy recovery within ten episodes, where the maximum value of scores is set as 500. As shown in the figure, our proposed policy-resilience mechanism achieves nearly the maximum value of scores in restoring policy performance, which presents a significant improvement over two baselines. Additionally, we observe that *Baseline:isolated policy resilience* performs worse than *Baseline:no policy resilience*, which indicates that recovered policies are adversely affected by locally fine-tuned dynamics models that do not accurately reflect deployment environments. We conclude from the comparison results that the design of our policy-resilience mechanism is capable of extracting and sharing critical knowledge about the environments, thus facilitating the development of an accurate dynamic model that generates imagined paths for effectively recovering the poisoned policy.

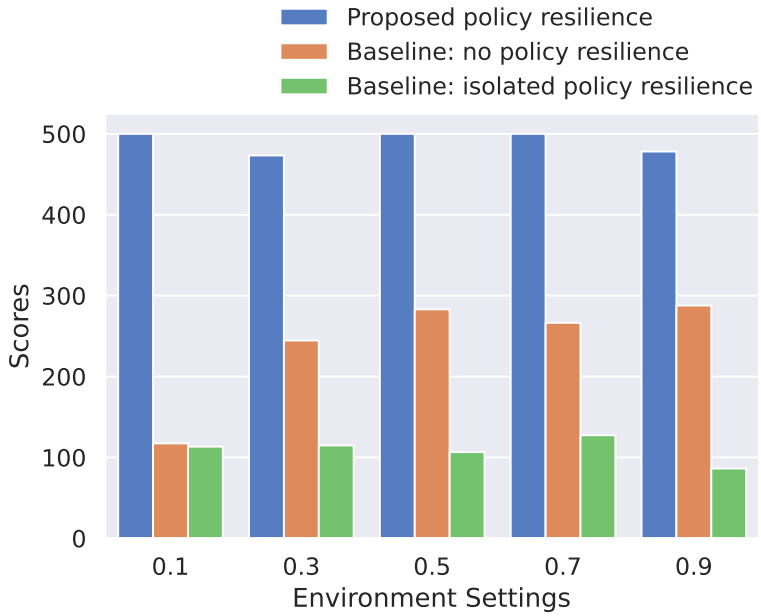


FIGURE 5.9: Comparison of policy-resilience performance on an *model-free* RL agent performing the Carpole task.

Model-based RL Agents. We have also performed similar experiments on federated model-based RL agents that are trained to control the Cartpole using MPC algorithms (see Chapter 5.4.3). Except for this difference, all other settings are the same as those used with model-free RL agents. Figure 5.10 illustrates the performance of policy recovery in deployment environments where pole lengths are set at $0.1 \sim 0.9$ (see Figure 5.8). As a result of adopting our policy-resilience mechanism, the agent achieves much higher scores than baselines in all deployment

environments, which indicates the success in recovering poisoned policies. Additionally, *Baseline:isolated policy resilience* performs better than *Baseline:no policy resilience* because of its locally updated dynamics model, however, it still performs poorly than our proposed approach particularly when training environments and deployment environments apparently varies in hyper-parameters. Consequently, an isolated update of the dynamics model cannot provide an accurate representation of the environment, resulting in somewhat limited policy recovery performance. In contrast, our policy-resilience mechanism encourages the sharing of critical environmental knowledge that facilitates the efficient development of an accurate dynamics model, which enables poisoned policies to be effectively recovered.

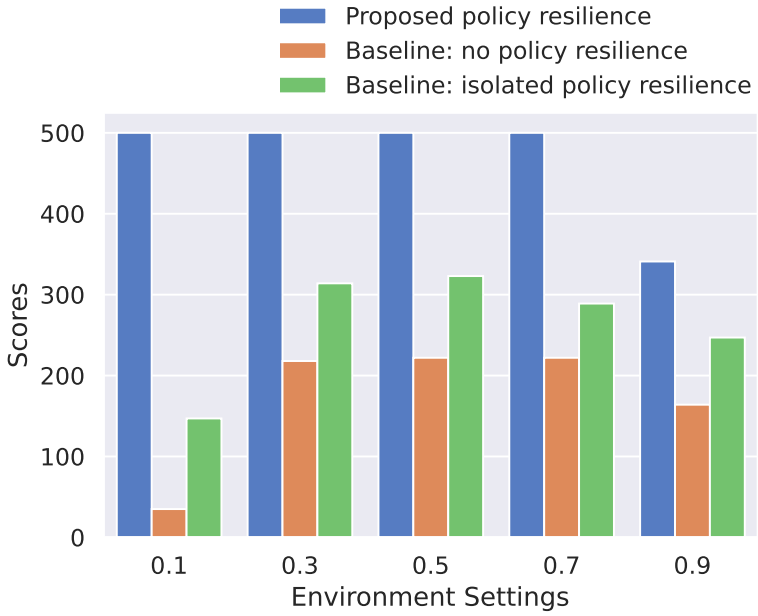


FIGURE 5.10: Comparison of policy-resilience performance on an *model-based* RL agent performing the Carpole task.

We also observe a significant difference between *Baseline:isolated policy resilience* in Figures 5.10 and 5.9, namely that locally updated dynamics models have a positive impact on policy recovery for model-based agents whereas they have a negative impact on policy recovery for model-free agents. This difference is explained by the fact that the model-free agent that uses MVE to recover its policy is more sensitive to the inaccuracy of the local dynamics model. As a result, this observation indirectly shows that the dynamics model learned by our policy-resilience mechanism is sufficiently accurate for the model-free agent to achieve the policy recovery objective.

5.6 Summary and Discussion

5.6.1 Summary

This chapter studies the protection of RL policies from the training-time environment-poisoning attack in terms of resilience. We propose a policy-resilience mechanism that attempts to recover poisoned policies for an optimal deployment performance. The policy-resilience mechanism is designed as a federated framework incorporated with a meta-learning approach, allowing efficient extraction and sharing of environment-structure information. Through the shared environment knowledge, the agent is able to grasp the dynamic of deployment environments and recover their policies in order to respond optimally to them. Such a policy-resilience mechanism is described as a three-step procedure, namely preparation, diagnosis and recovery. We empirically evaluate the policy resilience against TEPA in discrete state domains and against DBB-EPA in continuous state domains, showing the effectiveness and efficiency of the proposed policy-resilience mechanism.

5.6.2 Discussion

In this section, we provide detailed discussions of the assumptions and limitations of this work, including three points: the security issues of federated learning, the application scope of policy-resilience design, and the learning effectiveness of dynamics models.

Security Issues of Federated Framework. Throughout this work, we design the policy-resilience mechanism in a federated manner, because federated learning is a privacy-aware paradigm of model training. However, recent studies have indicated that FL does not always provide sufficient protection for privacy and robustness [110]. For example, it is possible that (1) an adversary server may attempt to gather sensitive information from individual updates over time, interfere with training procedures, or restrict participants' views of global parameters; (2) malicious participants may have the potential to disrupt the process of aggregating global parameters, poison the global model, or infer sensitive information about

other participants. Accordingly, the security issue of federated learning is a significant research topic that deserves further exploration, however, it is not the focus of our study. As a result, our policy-resilience mechanism is built on the assumption that the security of federated frameworks is guaranteed.

Applicability of Policy-Resilience Design. The applicability of our policy-resilience mechanism is discussed from two viewpoints: its potential limitations and its extended scope.

Our policy-resilience framework is constructed from a set of independent RL systems, which can result in restricted application if only one agent is present. Nevertheless, our proposed mitigation framework is applicable in many RL applications, such as robot cloud and robots [111], a state grid corporation and building grids [5], and a transport authority and autonomous vehicles [102]. Furthermore, as the Internet of Things (IoT) [112] and federated learning [100] advance, more isolated systems will be connected together for model learning or security protection, resulting in a wider application scope for our policy-resilience design.

We further claim that our policy-resilience approach can be applied to address policy adaptation problems, in which a learned RL policy is competent but specialized so that it will be ineffective in an unknown deployment environment. This problem is possible to be addressed by a classical meta-learning approach [104, 113]. However, the classical meta-learning approach assumes that the RL agent can access multiple environments throughout the training process, which is commonly impossible due to physical or security issues in real-world applications. For instance, when training a self-driving car in the tropics (e.g., Singapore), it is difficult to reproduce an icy road and snowy surroundings as a training environment. Therefore, this car would be at a loss in colder regions (e.g., Moscow) where such road conditions are natural. In this example, the physical issue prevents the RL agent from accessing various training environments. In light of this, agents, which can only learn their policy in a single training environment, should be able to quickly comprehend unknown deployment environments and accordingly improve policy performance. Our proposed policy-resilience mechanism, which incorporates a federated framework with a meta-learning mechanism, can fulfill this goal.

Learning Efficacy of Dynamics Model. In our work, the quality of the server dynamics model plays an influential role on the performance of policy recovery. As a result, the learning efficiency of the dynamics model is a key point in our policy-resilience mechanism. Dynamic modelling, however, may become more challenging as the environment becomes more complex. This may pose challenges to policy-resilience performance, which is a potential concern in our work. Nevertheless, this concern is expected to be addressed since a number of methods [114, 115] have been proposed to accurately represent the dynamics of the environment and these methods can be incorporated into our policy-resilience mechanisms.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this thesis, we focus on security issues of RL system from the perspective of training environment dynamics. We investigate environment poisoning attacks against RL in both white-box and black-box settings, and we study a policy-resilience mechanism to recover RL policies from such poisoning.

In chapter 3, we propose a transferable environment poisoning attack (TEPA) against RL at training time. We design the attack framework as a bi-level Markov Decision Process, and we learn an attack strategy in the white-box setting where the attacker has full knowledge of the agent's learning algorithm, policy model and its environment dynamics model. The attack strategy seeks adaptive and minimal environment changes that induce the agent to learn a target policy desired by the attacker. In addition, our attack strategy, which is learned on a white-box proxy agent, can be transfered to poison the policies of other agents that performs the same tasks but adopts different learning algorithms. In view of this transferability, we further investigate the environment-poisoning attacks against black-box agents and a population of independent agents. Compared to SOTA reward-poisoning attacks, TEPA shows comparable attack performance without assuming access to the victim's perception and memory, and it effectively poisons a variety of RL algorithms rather than targeting a specific one.

In chapter 4, we propose a double-black-box environment-poisoning attack (DBB-EPA), only assuming the attacker's ability to alter environment hyper-parameters. Considering that environment alteration comes at a cost, we seek minimal poisoning in an unknown environment and aim to force a black-box RL agent to learn an attacker-designed policy. To this end, we incorporate an inference module in our framework to capture the internal information of an unknown RL system and, accordingly, learn an adaptive strategy based on an approximation of our attack objective. The DBB-EPA has been shown to be effective against tabular-RL agents as well as deep-RL agents in discrete state as well as continuous state domains. As the DBB-EPA is independent of the victim's learning mechanism and environment model, it is a more realistic way of exposing the shortcomings of RL algorithms in poisoned environments.

In chapter 5, we propose a policy-resilience mechanism that enables an RL agent to recover its policy from environment-poisoning attacks. Policy resilience consists of three stages, namely preparation, diagnosis, and recovery. At the preparation stage, the critical knowledge of the deployment-environment structure is extracted, which is shared with the agent so that it can efficiently learn the dynamics model of the deployment environment at the diagnosis stage. During the recovery stage, the agent uses imagined trajectories derived from the dynamics model to restore the deployment capability of the poisoned policy. In the policy-resilience mechanism, the extraction and sharing of the environmental knowledge are achieved by design of a federated framework in conjunction with a meta-learning mechanism. Based on an empirical evaluation of our policy-resilience mechanism, we demonstrate that a poisoned policy can be restored to perform optimally during deployment. In light of the idea of knowledge sharing, the proposed policy-resilience mechanism is a resource-saving and time-efficient solution for the addressing harmful effects caused by environmental poisoning.

In summary, this thesis investigates the security issues arising from environment dynamics on the learning of RL policies in terms of poisoning and resilience, which contributes to the development of secure RL.

6.2 Future Directions

In the future, there may be four possible directions for further research, including the following:

- investigation of policy poisoning on multiple RL agents that interact in a common environment;
- development of attack generalization for multi-task RL agents;
- improvement of learning efficiency of an adaptive attack strategy;
- enhancement of protection solutions against environment-poisoning attacks.

We briefly describe each research direction in the follow.

Policy Poisoning in Multiple Agents. Currently, we have investigated the environment-poisoning attacks against a population of independent RL agents, attempting to induce a common attacker-desired policy throughout the population. Our future research will focus on multiple RL agents that interact in a common environment, such as soccer games and swarm robots. Instead of uniformly manipulating each agent's policy individually, we could manipulate the common environment to induce the whole team to reach an attacker-desired objectives. In addition, we may intentionally influence the learning of one agent's behaviour by using the relationships (e.g., cooperation or competition) or communications among teammates, and in this case, the teammates' behaviour is generally considered part of the agent's environment.

Attack against Multi-tasks Agents. Our environment-poisoning attack strategy has been demonstrated to be effective for various RL victims regardless of their learning algorithms, however, it is only effective for a specific RL task. Using a 3D grid world as an example, the attack strategy can successfully poison the victims who are performing the same navigation tasks as the proxy agent. In contrast, our future research attempts to generalize the attack strategy to a variety of RL tasks within the same environment. In such a scenario, the attacker-desired policy is not fixed and specific, rather it could become a behaviour rule. In the case of

3D navigation, we expect all victims can be poisoned by a single attack strategy, regardless of their starting point and destination, i.e., the strategy is effective for a variety of navigation tasks within the same environment.

Learning Efficiency of Attack Strategy. In current work, the learning efficiency of an attack strategy is a potential concern due to either the sample efficiency or the environment complexity.

First, the current attack strategy is learned using deep RL algorithms (e.g., DQN and DDPG) that requires a large number of samples. To avoid the sampling burden, it is possible to adopt model-based algorithms to train the attacker's strategy. Since model-based algorithms require dynamics models, the key challenge is the knowledge of the attacker's dynamics, i.e., how the victim updates policy given a poisoned training environment. The latent space model [116] is a potential solution to represent the attacker's dynamics system.

Second, learning an attack strategy is increasingly difficult as the RL agent's environment becomes more sophisticated, the attacker's action space becomes larger, or the attacker-desired behavior becomes more complex. To address this concern, we would adopt curriculum learning [58, 117] to efficiently learn attack strategy in the future work. For example, we could first split the attacker-desired behaviour into sub-behaviours and design a sequence of subsets automatically. Accordingly, the attacker could learn to enforce the sub-desired behaviour and transfer its attack skills along the sequence of target subsets, until it is able to enforce the whole attacker-desired behaviour on the victim.

Enhancement of Protection Solutions. In order to enhance protection solutions, there are three important directions that should be considered.

First, this thesis focuses on policy resilience against environment-poisoning attacks. Besides, it is also possible to apply the principle of our solution to the attack caused by perturbed reward functions. As an example, critical knowledge of reward functions can be extracted and shared within the federated framework, which allows an accurate evaluation of reward functions to lead to imagined trajectories for recovering policy performance. As such, in future work, we are going to empirically

investigate how policy-resilience mechanisms perform against reward-poisoning attacks during training.

Second, we currently assume that the security of the federated design can be guaranteed, so we can focus on investigating RL security issues. In the future, it is our intention to explicitly consider the potential security threats present in the federated framework and implement appropriate protection techniques.

Third, a policy-robustness solution against environmental poisoning would also be an imperative direction to pursue in the future. Note that a blind robustness to all changes in training environments is not an optimal strategy. This is primarily due to the fact that changing the training environment is a common method for improving the generalization of RL policies. As such, we expect the policy-robustness solution to be able to identify whether changes to the training environment are malicious or natural, and accordingly decide whether to remain robust or adapt.

List of Publications and Submissions

Workshops

- Hang Xu, Ridhima Bector, Zinovi Rabinovich, "Teaching Multiple Learning Agents by Environment-Dynamics Tweaks", Adaptive and Learning Agents Workshop at AAMAS, 2020.
- Hang Xu, Zinovi Rabinovich, "Truly Black-box Attack on Reinforcement Learning via Environment Poisoning", Adaptive and Learning Agents Workshop at AAMAS, 2021.
- **Hang Xu**, Xinghua Qu, Zinovi Rabinovich, "Reinforcement Learning Policy Resilience to Environment-Dynamics Poisoning", Machine Learning Security Workshop at NeurIPS, 2022.

Conferences

- **Hang Xu**, Rundong Wang, Lev Raizman, Zinovi Rabinovich, "Transferable environment poisoning: Training-time attack on reinforcement learning", Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'21), pp. 1398-1406, 2021.
- **Hang Xu**, Xinghua Qu, Zinovi Rabinovich, "Spiking Pitch Black: Poisoning an Unknown Environment to Attack Unknown Reinforcement Learners", Proceedings of the 21th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'22), pp. 1409-1417, 2022.

Journals

- **Hang Xu**, Xinghua Qu, Rundong Wang, Zinovi Rabinovich, "Transferable Environment Poisoning Attack against Reinforcement Learning", Journal of Artificial Intelligence (AIJ), 2022, pending submission.

Bibliography

[1] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. In *Proceedings of 28th British Machine Vision Conference*, pages 1–13, London, British, 2017. BMVA. 1

[2] 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*, 1(1):1–18, 2021. 11

[3] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017. 1

[4] Sunyong Kim and Hyuk Lim. Reinforcement learning based energy management algorithm for smart energy buildings. *Energies*, 11(8):2010–2029, 2018. 1

[5] Sangyoon Lee and Dae-Hyun Choi. Federated reinforcement learning for energy management of multiple smart homes with distributed energy resources. *IEEE Transactions on Industrial Informatics*, 18(1):488–497, 2022. 87, 107

[6] Tomah Sogabe, Dinesh Bahadur Malla, Shota Takayama, Seiichi Shin, Katsuyoshi Sakamoto, Koichi Yamaguchi, Thakur Praveen Singh, Masaru Sogabe, Tomohiro Hirata, and Yoshitaka Okada. Smart grid optimization by deep reinforcement learning over discrete and continuous action space. In *Proceedings of the 7th World Conference on Photovoltaic Energy Conversion*, pages 3794–3796, Hawaii, USA, 2018. IEEE. 1

[7] Elsa Riachi, Muhammad Mamdani, Michael Fralick, and Frank Rudzicz. Challenges for reinforcement learning in healthcare, 2021. 1

[8] Antonio Coronato, Muddasar Naeem, Giuseppe De Pietro, and Giovanni Paragliola. Reinforcement learning for intelligent healthcare applications: A survey. *Artificial Intelligence in Medicine*, 109:101964–101964, 2020. 1

[9] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *International Conference on Learning Representations Workshop*. ICLR, 2017. 1, 17, 18, 19, 20, 21, 28, 43

[10] 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 *The 28th International Joint Conference on Artificial Intelligence*. IJCAI, 2017. 17, 19, 20, 21

[11] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. In *International Conference on Learning Representations Workshop*. ICLR, 2017. 28, 30

[12] Edgar Tretschk, Seong Joon Oh, and Mario Fritz. Sequential attacks on agents for long-term adversarial goals. In *CSCS ACM Computer Science in Cars Symposium*, 2018. 18, 19, 20

[13] Matthew Inkawhich, Yiran Chen, and Hai Li. Snooping attacks on deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*. IFAAMS, 2019. 1, 28

[14] Yuzhe Ma, Xuezhou Zhang, Wen Sun, and Jerry Zhu. Policy poisoning in batch reinforcement learning and control. In *Advances in Neural Information Processing Systems*. ACM, 2019. 1, 3, 22, 24, 26, 27, 32, 37, 38, 43, 46, 49, 56

[15] Xuezhou Zhang, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. Adaptive reward-poisoning attacks against reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020. 22, 25, 27, 38, 49, 53, 54, 56

[16] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020. 3, 22, 26, 27, 29, 32, 37, 43, 46, 49, 64

[17] Amin Rakhsha, Xuezhou Zhang, Xiaojin Zhu, and Adish Singla. Reward poisoning in reinforcement learning: Attacks against unknown learners in unknown environments. In *Proceedings of the 35th Conference on Neural Information Processing Systems*, pages 1–22, Online, 2021. ACM. 22, 25, 28, 49

[18] Yanchao Sun, Da Huo, and Furong Huang. Vulnerability-aware poisoning mechanism for online rl with unknown dynamics. In *Proceedings of 10th International Conference on Learning Representation*, pages 1–27, Vienna, Austria, 2021. ICLR. 1, 22, 23, 28, 49

[19] Gina Neff. Talking to bots: Symbiotic agency and the case of tay. *International Journal of Communication*, 2016. 1

[20] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust

physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018. 1

[21] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *Proceedings of 6th International Conference on Learning Representations*, pages 1–17, Vancouver, Canada, 2018. ICLR. 2, 29

[22] Christian Perez, Felipe Petroski Such, and Theofanis Karaletsos. Generalized hidden parameter mdps: Transferable model-based rl in a handful of trials. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 5403–5411, New York, USA, 2020. AAAI. 2, 86

[23] Sumedh A Sontakke, Arash Mehrjou, Laurent Itti, and Bernhard Schölkopf. Causal curiosity: Rl agents discovering self-supervised experiments for causal representation learning. In *Proceedings of the 38th International Conference on Machine Learning*, pages 9848–9858, Online, 2021. PMLR. 2, 29

[24] Taylor W Killian, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Proceedings of the 31th AAAI Conference on Artificial Intelligence*, pages 4949–4950, California, USA, 2017. AAAI. 2, 84

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 10, 12, 26, 37, 46, 53, 56, 74, 76

[26] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 11

[27] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274, 2013. 11

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Proceedings of the 27th Conference on Neural Information Processing Systems*, pages 1–9, USA, 2013. ACM. 11, 76

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. 13, 77

[30] 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 *Proceedings of 4th International*

Conference on Learning Representation, pages 1–14, San Juan, Puerto Rico, 2016. ICLR. 14, 52, 72

[31] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*. PMLR, 10–15 Jul 2018. 14, 45, 54, 76

[32] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning*. PMLR, 2012. 16

[33] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013. 16

[34] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016. 17, 28

[35] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proceding of International Conference on Autonomous Agents and Multiagent Systems*, 2017. 17, 19, 20, 30

[36] Xinlei Pan, Chaowei Xiao, Warren He, Shuang Yang, Jian Peng, Mingjie Sun, Mingyan Liu, Bo Li, and Dawn Song. Characterizing attacks on deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2022. 17, 19, 20

[37] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. 18, 19

[38] Shumeet Baluja and Ian Fischer. Learning to attack: Adversarial transformation networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI press, 2018. 18, 20

[39] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015. 19

[40] 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. PMLR, 2016. 19

[41] 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*. Springer, 2017. 22, 28, 37

[42] Yi Han, Benjamin IP Rubinstein, Tamas Abraham, Tansu Alpcan, Olivier De Vel, Sarah Erfani, David Hubczenko, Christopher Leckie, and Paul Montague. Reinforcement learning for autonomous defence in software-defined networking. In *International Conference on Decision and Game Theory for Security*. Springer, 2018. 22

[43] Léonard Hussenot, Matthieu Geist, and Olivier Pietquin. Copycat: Taking control of neural policies with constant attacks. In *International Conference on Autonomous Agents and Multiagent Systems*, 2020. 22, 23, 28

[44] Xian Yeow Lee, Sambit Ghadai, Kai Liang Tan, Chinmay Hegde, and Soumik Sarkar. Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI press, 2020. 22, 23

[45] Yunhan Huang and Quanyan Zhu. Deceptive reinforcement learning under adversarial manipulations on cost signals. In *International Conference on Decision and Game Theory for Security*. Springer, 2019. 22, 24

[46] Xiaoxuan Bai, Wenjia Niu, Jiqiang Liu, Xu Gao, Yingxiao Xiang, and Jingjing Liu. Adversarial examples construction towards white-box q table variation in dqn pathfinding training. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018. 22, 25, 64

[47] Tong Chen, Wenjia Niu, Yingxiao Xiang, Xiaoxuan Bai, Jiqiang Liu, Zhen Han, and Gang Li. Gradient band-based adversarial training for generalized attack immunity of a3c path finding. *CoRR*, abs/1807.06752, 2018. 22, 26

[48] Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. *Advances in neural information processing systems*, 19, 2006. 26

[49] David Isele, Mohammad Rostami, and Eric Eaton. Using task features for zero-shot knowledge transfer in lifelong learning. In *Ijcai*, volume 16, pages 1620–1626, 2016. 27

[50] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*, pages 316–324. PMLR, 2014. 27

[51] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI press, 2017. 29

[52] Sarah Keren, Luis Pineda, Avigdor Gal, Erez Karpas, and Shlomo Zilberstein. Equi-reward utility maximizing design in stochastic environments. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. IJCAI, 2017. doi: 10.24963/ijcai.2017/608. 29

[53] Zinovi Rabinovich, Lachlan Dufton, Kate Larson, and Nick Jennings. Cultivating desired behaviour: Policy teaching via environment-dynamics tweaks. In *Proceedings of International Conference on Autonomous Agents and MultiAgent Systems*. IFAAMS, 2010. 39, 40, 44, 52, 66, 72

[54] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI, 2020. 29

[55] Jette Randløv. Shaping in reinforcement learning by changing the physics of the problem. In *Proceedings of the 27th International Conference on Machine Learning*. PMLR, 2000. 29

[56] Daniel S Brown and Scott Niekum. Machine teaching for inverse reinforcement learning: Algorithms and applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33. AAAI press, 2019. 29

[57] Kamalaruban Parameswaran, Rati Devidze, Volkan Cevher, and Adish Singla. Interactive teaching algorithms for inverse reinforcement learning. In *The 28th International Joint Conference on Artificial Intelligence*. IJCAI, 2019. 29

[58] Sanmit Narvekar and Peter Stone. Learning curriculum policies for reinforcement learning. In *Proceedings of International Conference on Autonomous Agents and MultiAgent Systems*, 2018. 29, 112

[59] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in Neural Information Processing Systems*, 33, 2020. 29

[60] Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala Al-Fuqaha, Dinh Thai Huang, and Dusit Niyato. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE Transactions on Artificial Intelligence*, 1(1):1–21, 2021. 30

[61] Huan Zhang, Hongge Chen, Duane Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. *arXiv preprint arXiv:2101.08452*, 2021. 30, 34, 83

[62] Tuomas Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. Robust deep reinforcement learning through adversarial loss. *Advances in Neural Information Processing Systems*, 34:26156–26167, 2021. 30

[63] 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. *Advances in Neural Information Processing Systems*, 33:21024–21037, 2020. 30

[64] 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. 30

[65] Yen-Chen Lin, Ming-Yu Liu, Min Sun, and Jia-Bin Huang. Detecting adversarial attacks on neural network policies with visual foresight. *arXiv preprint arXiv:1710.00814*, 2017. 31

[66] Victor Gallego, Roi Naveiro, and David Rios Insua. Reinforcement learning under threats. *arXiv preprint arXiv:1809.01560*, 2018.

[67] Yingxiao Xiang, Wenjia Niu, Jiqiang Liu, Tong Chen, and Zhen Han. A pca-based model to predict adversarial examples on q-learning of path finding. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 773–780. IEEE Computer Society, 2018. 31

[68] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015. 31

[69] Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1331–1340. PMLR, 2019. 31

[70] Mario Bravo and Panayotis Mertikopoulos. On the robustness of learning in games with stochastically perturbed payoff observations. *Games and Economic Behavior*, 103:41–66, 2017. 31

[71] Olalekan Ogunmolu, Nicholas Gans, and Tyler Summers. Minimax iterative dynamic game: Application to nonlinear robot control tasks. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6919–6925. IEEE, 2018. 31

[72] Mohammed Amin Abdullah, Hang Ren, Haitham Bou Ammar, Vladimir Milenkovic, Rui Luo, Mingtian Zhang, and Jun Wang. Wasserstein robust reinforcement learning. *arXiv preprint arXiv:1907.13196*, 2019. 31

[73] Vahid Behzadan and Arslan Munir. Whatever does not kill deep reinforcement learning, makes it stronger. *arXiv preprint arXiv:1712.09344*, 2017. 31, 32, 34, 83, 84

[74] Vahid Behzadan and Arslan Munir. Mitigation of policy manipulation attacks on deep q-networks with parameter-space noise. In *International Conference on Computer Safety, Reliability, and Security*, pages 406–417. Springer, 2018. 31, 32, 34

[75] Kiarash Banihashem, Adish Singla, and Goran Radanovic. Defense against reward poisoning attacks in reinforcement learning. *arXiv preprint arXiv:2102.05776*, 2021. 31, 32, 34

[76] Jingkang Wang, Yang Liu, and Bo Li. Reinforcement learning with perturbed rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6202–6209, 2020. 31, 33, 34

[77] Thodoris Lykouris, Max Simchowitz, Alex Slivkins, and Wen Sun. Corruption-robust exploration in episodic reinforcement learning. In *Conference on Learning Theory*, pages 3242–3245. PMLR, 2021. 31, 33, 34

[78] Yifang Chen, Simon Du, and Kevin Jamieson. Improved corruption robust algorithms for episodic reinforcement learning. In *International Conference on Machine Learning*, pages 1561–1570. PMLR, 2021. 31, 33

[79] Chen-Yu Wei, Christoph Dann, and Julian Zimmert. A model selection approach for corruption robust reinforcement learning. In *International Conference on Algorithmic Learning Theory*, pages 1043–1096. PMLR, 2022. 31, 33

[80] Xuezhou Zhang, Yiding Chen, Xiaojin Zhu, and Wen Sun. Robust policy gradient against strong data corruption. In *International Conference on Machine Learning*, pages 12391–12401. PMLR, 2021. 31, 33, 34, 83

[81] Xuezhou Zhang, Yiding Chen, Xiaojin Zhu, and Wen Sun. Corruption-robust offline reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 5757–5773. PMLR, 2022. 31, 34

[82] Fan Wu, Linyi Li, Chejian Xu, Huan Zhang, Bhavya Kailkhura, Krishnaram Kenthapadi, Ding Zhao, and Bo Li. Copa: Certifying robust policies for offline reinforcement learning against poisoning attacks. *arXiv preprint arXiv:2203.08398*, 2022. 31, 34, 83

[83] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015. 35

[84] Philip E Agre. The dynamic structure of everyday life. Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1988. 37

[85] Haoqi Zhang and David C Parkes. Value-based policy teaching with active indirect elicitation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 8. AAAI press, 2008. 38

[86] Haoqi Zhang, David C Parkes, and Yiling Chen. Policy teaching through reward function learning. In *Proceedings of the 10th ACM conference on Electronic commerce*. ACM, 2009. 38, 76

[87] Ziad Rached, Fady Alajaji, and L Lorne Campbell. The kullback-leibler divergence rate between markov sources. *IEEE Transactions on Information Theory*, 50(5):917–921, 2004. 38, 40, 41

[88] 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):529–533, 2015. 45

[89] Aditya Grover, Maruan Al-Shedivat, Jayesh Gupta, Yuri Burda, and Harrison Edwards. Learning policy representations in multiagent systems. In Jennifer Dy and Andreas Krause, editors, *Thirty-fifth International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*. PMLR, 10–15 Jul 2018. 47

[90] Rundong Wang, Runsheng Yu, Bo An, and Zinovi Rabinovich. I2hrl: Interactive influence-based hierarchical reinforcement learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI, 2020. 47

[91] Ali Eslami, Farzaneh Abdollahi, and Khashayar Khorasani. Stochastic fault and cyber-attack detection and consensus control in multi-agent systems. *International Journal of Control*, 0(0):1–19, 2021. doi: 10.1080/00207179.2021.1912394. URL `https://doi.org/10.1080/00207179.2021.1912394`. 49

[92] Aquib Mustafa and Hamidreza Modares. Attack analysis and resilient control design for discrete-time distributed multi-agent systems. *IEEE Robotics and Automation Letters*, 5(2):369–376, 2019. 49

[93] In-Sun Choi, Junho Hong, and Tae-Wan Kim. Multi-agent based cyber attack detection and mitigation for distribution automation system. *IEEE Access*, 8:183495–183504, 2020. 49

[94] Xinlei Pan, Weiyao Wang, Xiaoshuai Zhang, Bo Li, Jinfeng Yi, and Dawn Song. How you act tells a lot: Privacy-leaking attack on deep reinforcement learning. In *Proceedings of International Conference on Autonomous Agents and MultiAgent Systems*. IFAAMS, 2019. 52

[95] Hang Xu, Rundong Wang, Lev Raizman, and Zinovi Rabinovich. Transferable environment poisoning: Training-time attack on reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1398–1406, Online, 2021. IFAAMAS. 64, 72

[96] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017. 70

[97] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 75

[98] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. 80

[99] Finale Doshi-Velez and George Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *IJCAI: proceedings of the conference*, volume 2016, page 1432. NIH Public Access, 2016. 86

[100] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. 86, 97, 101, 107

[101] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019. 87

[102] Xinle Liang, Yang Liu, Tianjian Chen, Ming Liu, and Qiang Yang. Federated transfer reinforcement learning for autonomous driving. *arXiv preprint arXiv:1910.06001*, 2019. 87, 107

[103] Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, and Youn-Hee Han. Federated reinforcement learning for training control policies on multiple iot devices. *Sensors*, 20(5):1359, 2020. 87

[104] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *International Conference on Machine Learning (ICML)*, 2017. URL `http://arxiv.org/abs/1703.03400`. 87, 90, 91, 107

[105] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016. 95

[106] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017. 95

[107] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018. 95

[108] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31, 2018. 95

[109] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005. 95

[110] Lingjuan Lyu, Han Yu, Xingjun Ma, Lichao Sun, Jun Zhao, Qiang Yang, and Philip S Yu. Privacy and robustness in federated learning: Attacks and defenses. *arXiv preprint arXiv:2012.06337*, 2020. 106

[111] Boyi Liu, Lujia Wang, and Ming Liu. Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems. *IEEE Robotics and Automation Letters*, 4(4):4555–4562, 2019. 107

[112] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information systems frontiers*, 17(2):243–259, 2015. 107

[113] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019. 107

[114] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018. 108

[115] Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K Hubert, and David Silver. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2021. 108

[116] Daniel K Sewell and Yuguo Chen. Latent space models for dynamic networks. *Journal of the American Statistical Association*, 110(512):1646–1657, 2015. 112

[117] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017. 112