Policy Resilience to Environment Poisoning Attacks on Reinforcement Learning

Hang Xu
School of Computer Science and Engineering
Nanyang Technological University, Singapore
hang017@e.ntu.edu.sg

Xinghua Qu
ByteDance AI Lab
Singapore
quxinghua17@gmail.com

Zinovi Rabinovich
School of Computer Science and Engineering
Nanyang Technological University, Singapore
zinovi@ntu.edu.sg

Abstract

This paper investigates policy resilience to training-environment poisoning attacks on reinforcement learning (RL) policies, with the goal of recovering the deployment performance of a poisoned RL policy. Due to the fact that the policy resilience is an add-on concern to RL algorithms, it should be resource-efficient, time-conserving, and widely applicable without compromising the performance of RL algorithms. This paper proposes such a policy-resilience mechanism based on an idea of knowledge sharing. We summarize the policy resilience as three stages: preparation, diagnosis, recovery. Specifically, we design the mechanism as a federated architecture coupled with a meta-learning manner, pursuing an efficient extraction and sharing of the environment knowledge. With the shared knowledge, a poisoned agent can quickly identify the deployment condition and accordingly recover its policy performance. We empirically evaluate the resilience mechanism for both model-based and model-free RL algorithms, showing its effectiveness and efficiency in restoring the deployment performance of a poisoned policy.

1 Introduction

The security of Reinforcement Learning (RL) becomes increasingly significant as RL systems have been widely adopted in real-world applications [1–9]. Thus, it is essential to protect RL-based applications against a variety of adversarial attacks [10–14]. Among these attacks, environment poisoning attacks (EPAs) [15, 16] are considered particularly insidious. They poison RL policies by perturbing the training-environment *causal factors* (i.e., *hyper-parameters*), such as surface frictions, which are exposed outside of RL agents and easily accessible to third parties. In this paper, we attempt to address the security threat of EPAs from the perspective of resilience. The resilience, in this context, refers to an RL agent's ability to recover its policy from malicious manipulations [17].

In existing EPAs [15, 16], the dynamics of training environment are manipulated as a function of hyper-parameters. In other words, the poisoned environment retains the same environment structure (e.g., function) as the natural one, but differs in hyper-parameters. As a result, if an RL agent is equipped with the knowledge of environment structures and the capability of identifying hyper-parameters, it will be able to grasp the environment dynamics more efficiently than an agent that learns from scratch. Accordingly, this paper designs a resource- and time-efficient policy resilience by exploiting the knowledge of environment structure.

ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

In this paper, the proposed policy resilience can be summarized as three stages: preparation, diagnosis and recovery. First, the preparation stage intends to extract the critical knowledge of environment structures during the training of RL policies. This is achieved based on a design of federated framework incorporating with a meta-learning approach. Then, the diagnosis stage occurs prior to the deployment of the learned policy. At this stage, the RL agent is shared with the environment knowledge so that it efficiently identifies the dynamics of its deployment environment. At the recovery stage, the agent imagines trajectories based on its accurate understanding of the environment dynamics. Accordingly it recovers the poisoned policy for an optimal deployment performance. Furthermore, it is important to note that the proposed mechanism organizes independent and isolated RL systems in a federated manner, which enables the extraction and sharing of environment knowledge. The federated design is common in RL-based applications. For example, autonomous vehicles could be organized by a transportation agency in a federated manner, as shown in Figure 1. The transportation agency (i.e., server) can collect road conditions (e.g., client information) from multiple vehicles and share the merged information to each. In this way, it could assist a single vehicle to drive well 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. These examples imply our proposed policy-resilience mechanism can be applicable to many real-world scenarios.

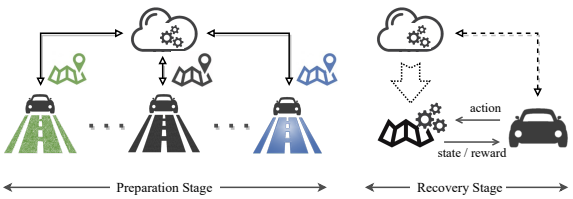


Figure 1: Example of policy-resilience mechanism.

As a summary, this paper makes the following contributions:

- We propose a policy-resilience mechanism specifically for environment poisoning attacks [15, 16] on RL. 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 can be described as a three-step procedure that includes preparation, diagnosis and recovery. Meta-learning approach is incorporated along with federated manner in the stage preparation and diagnosis, to ensure an efficient extraction of environment knowledge and an accurate interpretation of deployment dynamics.
- The policy-resilience mechanism has been evaluated empirically on both model-free and model-based RL agents, which demonstrate that poisoned policies can be recovered in a resource-efficient and time-efficient manner.

2 Problem Statement

Description of Environment Poisoning Attacks: Our policy-resilience mechanism targets the environment-poisoning attack [15, 16] against RL at training-time. The attack objective is to force an RL agent (i.e., victim) to learn an attacker-desired policy via minimized changes to the training environment. Specifically, at intervals of the victim's learning process, the attacker manipulates the victim's environment hyper-parameters responding to the victim's behaviour policy and training-environment conditions, until the victim obtains the policy designed by the attacker.

Formulation of Policy Resilience: Our policy-resilience mechanism organizes multiple independent and isolated RL systems in a federated manner. As shown in Figure 1, the federated policy-resilience framework consists of one server and a set of client RL systems, each with learnable dynamics models. We formalize the framework as $\langle \mathcal{S}, \{\mathcal{C}_i\}_{i=1}^{K} \rangle$. $\mathcal{S}$ represents a dynamics model in the reliable server, which is a function parameterized by $\theta_{\mathcal{S}}$. $\{\mathcal{C}_i\}_{i=1}^{K}$ represents the set of client RL systems governed by the server, where K is the number of systems. Within each of these RL systems, the environment should share a common underlying structure and could differ in hyper-parameters. Specifically, each client system is described as a tuple $\mathcal{C}_i = \langle \mathcal{R}_i, \mathcal{M}_i, \mathcal{D}_i, \mathcal{T}_i \rangle$. Here, $\mathcal{R}_i$ 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 $\langle s, a, F_{e_i}, r, \gamma \rangle$ (refer to Appendix B). Note that $\{\mathcal{M}_i\}_{i=1}^{K}$ belong to the same Hidden Parameter Markov Decision Process

(Hip-MDP) [18] with hyper-parameters $e_i \sim p(e)$ which parameterize the dynamics function F_{e_i}. $\mathcal{D}_i$ is a local dynamics model that shares the same structure as the server one $\mathcal{S}$. And $\mathcal{T}_i$ is a buffer that stores the RL agent's transitions $\langle s, a, s' \rangle$ used for learning a local dynamics model $\mathcal{D}_i$.

Importantly, we assume that the security of the federated framework is guaranteed. This assumption allows us to focus on the security threats occurring within a single isolated RL system. Additionally, it is forbidden for each RL agent to directly interact with other agents' environments or access raw transitions in others' buffers. Instead, clients share local information with the server and trust the knowledge retrieved from the server.

3 Methodology

In this section, we introduce our policy-resilience mechanism in accordance with the procedure, namely, preparation, diagnosis and recovery. First, we describe mathematically how the environment knowledge is acquired using a meta-learning approach within the federated framework (i.e., the preparation stage). After that, we describe how the poisoned policy can be recovered in response to an accurate understanding of the deployment environment (i.e., the diagnosis and recovery stages).

Preparation: Preparation occurs during the training of RL policies, aiming to efficiently extract critical knowledge about the environment structure. We utilize a meta-learning approach [19] to learn the server dynamics model $\mathcal{S}$ based on environment information collected from all the federated RL systems. Here, $\mathcal{S}$ is learned as a parameterized function, which can be used to quickly model environments that share a common structure but differ in hyper-parameters. As shown in Figure 5 in Appendix C, the learning of $\mathcal{S}$ 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). Such a process whereby the server transmits information to client systems and then receives messages from them is regarded as a round.

Mathematically, at the first step of each round, local dynamics models $\{\mathcal{D}_i\}_{i=1}^{K}$ should be initialized with the parameters $\theta_{\mathcal{S}}$ which are retrieved from the server one $\mathcal{S}$. After the initialization, loss values are locally calculated in a meta-learning way [19]. In more detail, in each client RL system where the 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 is 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 $\mathcal{T}_{e_i}^{qry} = \{\langle \bar{s}_n, \bar{a}_n, \bar{s}'_n \rangle\}_{n=1}^{N}$. Afterward, 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, resulting in updated parameters $\theta_i^{'}$. Then, $\mathcal{D}_{\theta_i^{'}}$ is evaluated on the query set $\mathcal{T}_{e_i}^{qry}$, and a loss value $\mathcal{L}(\mathcal{T}_{e_i}^{qry}, \mathcal{D}_{\theta_i^{'}})$ is calculated as a consequence. Note that this loss value can reflect the capability of identifying the hyper-parameters, i.e., the learning ability of $\mathcal{D}_{\theta_i=\theta_{\mathcal{S}}}$. At the third and the forth step, loss values $\left\{\mathcal{L}(\cdot, \theta_i^{'})\right\}_{i=1}^{K}$ are collected by the server from all the federated RL systems, which are utilized for optimizing the server dynamics model $\mathcal{S}(\theta_{\mathcal{S}})$ as the following objective:

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

where α is the learning rate. Here, the loss function $\mathcal{L}$ can be defined as Mean Square Error (MSE) in continuous state domains or as Cross Entropy in discrete state domains. When the parameters $\theta_{\mathcal{S}}$ are optimized, the server dynamics model $\mathcal{S}$ is a function parameterized by $\theta_{\mathcal{S}}^{*}$. $\mathcal{S}(\theta_{\mathcal{S}}^{*})$ is considered representative knowledge of the training-environment structure, capable of quickly personalizing to an environment with specific hyper-parameters.

Diagnosis: Diagnosis is a time-sensitive and resource-sensitive stage, which is carried out prior to the deployment of the learned policy. Namely, a poisoned agent is expected to quickly understand the dynamics of its deployment environment based on a small number of interactions. To achieve this, the agent is provided with the critical knowledge of environment structure and accordingly identify the environment dynamics. Such a diagnosis stage consists of three steps: (1) the retrieval of knowledge (i.e., parameters) from the server dynamics model; (2) the collection of trajectories in the deployment environment; (3) the understanding of the deployment environment (i.e., dynamics model).

Imagine an RL agent which has learned its policies in a poisoned training environment and aims to accurately grasp the dynamics of the deployment environment parameterized by $\hat{e} \sim p(e)$. The agent first initializes its local dynamics model using the parameters $\theta^*_{\mathcal{S}}$ retrieved from the server dynamics model $\mathcal{S}$. Then, the local dynamics model $\mathcal{D}_{\theta_{\hat{e}}=\theta^*_{\mathcal{S}}}$ is 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, denoted as

$$\theta'_{\hat{e}} = \theta_{\hat{e}} - \alpha \nabla_{\theta_{\hat{e}}} \mathcal{L}(\mathcal{T}_{\hat{e}}, \mathcal{D}_{\theta_{\hat{e}}=\theta^*_{\mathcal{S}}}). \tag{2}$$

Note that 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 $\theta^*_{\mathcal{S}}$.

Recovery: Recovery is the final stage of the policy-resilience mechanism, which aims to reduce the negative effects of the EPA and restore policy deployment performance to the full extent of its ability. The policy recovery is accomplished using the imagined trajectories derived from the dynamics model $\mathcal{D}_{\theta'_{\hat{e}}}$ that has been tailored to the deployment environment. The success of recovery are generally applicable for both model-free and model-based RL agent. For example, a model-free (MF) RL agent can recover its policy by adopting model-based value expansion [20, 21] while a model-based (MB) RL agent may choose Model Predictive Control with cross-entropy method [22] for restoring its policy.

4 Experiments

We evaluate the recovery of a poisoned RL policy in a 3D grid world, when different proportions of poisoned agents are present at the preparation stage. Figure 2 shows that poisoned policies are effectively recovered to good performance within only two-episode interactions. This means that the policy recovery is time- and resource-efficient due to the knowledge sharing. We further evaluate the applicability of our policy-resilience mechanism, showing the successful policy recovery on agents adopting either model-free (MF) or model-based (MB) RL learning algorithms, as shown in Figure 9 and 10. More results and discussions refer to Appendix E.

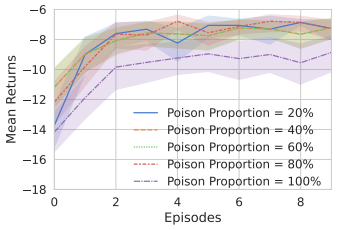


Figure 2: 3D Grid world

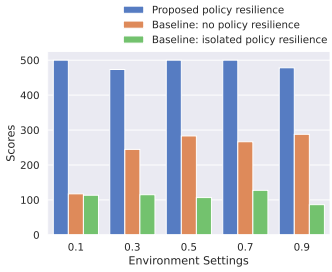


Figure 3: Cartpole, MF

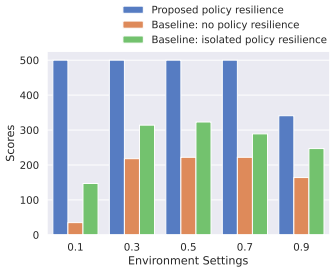


Figure 4: Cartpole, MB

5 Conclusion

The aim of this paper is to investigate resilience of RL policies against the training-time environment poisoning attacks. A policy-resilience mechanism is proposed to recover a poisoned policy for an optimal deployment performance, which is described as preparation, diagnosis and recovery. It is designed as a federated framework incorporated with a meta-learning approach, allowing efficient extraction and sharing of environment knowledge. When armed with shared knowledge, a poisoned agent can effectively identify the dynamics model of the deployment environment from limited interactions. In this way, imagined trajectories can be derived from the dynamics model, which is then used to recover the deployment performance of the poisoned policy. Our empirical evaluation shows the effectiveness and efficiency of our policy resilience to EPA on both model-free and model-based RL agents.

References

[1] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proceedings of 28th British Machine Vision Conference*. London, British: BMVA, 2017, pp. 1–13.

[2] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[3] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 1, pp. 1–18, 2021.

[4] S. Kim and H. Lim, "Reinforcement learning based energy management algorithm for smart energy buildings," *Energies*, vol. 11, no. 8, pp. 2010–2029, 2018.

[5] T. Sogabe, D. B. Malla, S. Takayama, S. Shin, K. Sakamoto, K. Yamaguchi, T. P. Singh, M. Sogabe, T. Hirata, and Y. 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*. Hawaii, USA: IEEE, 2018, pp. 3794–3796.

[6] S. Lee and D.-H. Choi, "Federated reinforcement learning for energy management of multiple smart homes with distributed energy resources," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 1, pp. 488–497, 2022.

[7] N. Eghbali, T. Alhanai, and M. M. Ghassemi, "Patient-specific sedation management via deep reinforcement learning," *Frontiers in Digital Health*, vol. 3, pp. 1–9, 2021.

[8] A. Coronato, M. Naeem, G. De Pietro, and G. Paragliola, "Reinforcement learning for intelligent healthcare applications: A survey," *Artificial Intelligence in Medicine*, vol. 109, pp. 101 964–101 964, 2020.

[9] E. Riachi, M. Mamdani, M. Fralick, and F. Rudzicz, "Challenges for reinforcement learning in healthcare," 2021.

[10] Y. Ma, X. Zhang, W. Sun, and J. Zhu, "Policy poisoning in batch reinforcement learning and control," in *Advances in Neural Information Processing Systems*. ACM, 2019.

[11] X. Zhang, Y. Ma, A. Singla, and X. Zhu, "Adaptive reward-poisoning attacks against reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020.

[12] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," in *International Conference on Learning Representations Workshop*. ICLR, 2017.

[13] Y.-C. Lin, M.-Y. Liu, M. Sun, and J.-B. Huang, "Detecting adversarial attacks on neural network policies with visual foresight," *arXiv preprint arXiv:1710.00814*, 2017.

[14] A. Rakhsha, G. Radanovic, R. Devidze, X. Zhu, and A. 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.

[15] H. Xu, R. Wang, L. Raizman, and Z. Rabinovich, "Transferable environment poisoning: Training-time attack on reinforcement learning," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. Online: IFAAMAS, 2021, pp. 1398–1406.

[16] H. Xu, X. Qu, and Z. Rabinovich, "Spiking pitch black: Poisoning an unknown environment to attack unknown reinforcement learners," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1409–1417.

[17] V. Behzadan and A. Munir, "Whatever does not kill deep reinforcement learning, makes it stronger," *arXiv preprint arXiv:1712.09344*, 2017.

[18] T. W. Killian, G. Konidaris, and F. Doshi-Velez, "Robust and efficient transfer learning with hidden parameter markov decision processes," in *Proceedings of the 31th AAAI Conference on Artificial Intelligence*. California, USA: AAAI, 2017, pp. 4949–4950.

[19] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," *International Conference on Machine Learning (ICML)*, 2017. [Online]. Available: http://arxiv.org/abs/1703.03400

[20] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," *arXiv preprint arXiv:1803.00101*, 2018.

[21] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," *Advances in neural information processing systems*, vol. 31, 2018.

[22] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[23] K. Banihashem, A. Singla, and G. Radanovic, "Defense against reward poisoning attacks in reinforcement learning," *arXiv preprint arXiv:2102.05776*, 2021.

[24] T. Lykouris, M. Simchowitz, A. Slivkins, and W. Sun, "Corruption-robust exploration in episodic reinforcement learning," in *Conference on Learning Theory*. PMLR, 2021, pp. 3242–3245.

[25] Y. Chen, S. Du, and K. Jamieson, "Improved corruption robust algorithms for episodic reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1561–1570.

[26] C.-Y. Wei, C. Dann, and J. Zimmert, "A model selection approach for corruption robust reinforcement learning," in *International Conference on Algorithmic Learning Theory*. PMLR, 2022, pp. 1043–1096.

[27] H. Zhang, H. Chen, D. Boning, and C.-J. Hsieh, "Robust reinforcement learning on state observations with learned optimal adversary," *arXiv preprint arXiv:2101.08452*, 2021.

[28] X. Zhang, Y. Chen, X. Zhu, and W. Sun, "Robust policy gradient against strong data corruption," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 391–12 401.

[29] F. Wu, L. Li, C. Xu, H. Zhang, B. Kailkhura, K. Kenthapadi, D. Zhao, and B. Li, "Copa: Certifying robust policies for offline reinforcement learning against poisoning attacks," *arXiv preprint arXiv:2203.08398*, 2022.

[30] V. Behzadan and A. Munir, "Mitigation of policy manipulation attacks on deep q-networks with parameter-space noise," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 406–417.

[31] J. Wang, Y. Liu, and B. Li, "Reinforcement learning with perturbed rewards," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 6202–6209.

[32] F. Doshi-Velez and G. Konidaris, "Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations," in *IJCAI: proceedings of the conference*, vol. 2016. NIH Public Access, 2016, p. 1432.

[33] C. Perez, F. P. Such, and T. Karaletsos, "Generalized hidden parameter mdps: Transferable model-based rl in a handful of trials," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. New York, USA: AAAI, 2020, pp. 5403–5411.

[34] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[35] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[36] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated transfer reinforcement learning for autonomous driving," *arXiv preprint arXiv:1910.06001*, 2019.

[37] H.-K. Lim, J.-B. Kim, J.-S. Heo, and Y.-H. Han, "Federated reinforcement learning for training control policies on multiple iot devices," *Sensors*, vol. 20, no. 5, p. 1359, 2020.

[38] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[39] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[40] L. Lyu, H. Yu, X. Ma, L. Sun, J. Zhao, Q. Yang, and P. S. Yu, "Privacy and robustness in federated learning: Attacks and defenses," *arXiv preprint arXiv:2012.06337*, 2020.

[41] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019.

[42] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information systems frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[43] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson, "Varibad: A very good method for bayes-adaptive deep rl via meta-learning," *arXiv preprint arXiv:1910.08348*, 2019.

[44] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in neural information processing systems*, vol. 31, 2018.

[45] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver, "Planning in stochastic environments with a learned model," in *International Conference on Learning Representations*, 2021.

[46] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver, "Online and offline reinforcement learning by planning with a learned model," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 580–27 591, 2021.

A Related Works

To guarantee the security associated with the learning of RL policies, defences 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 [17]. There are two general categories of these works: (1) Some works [23–29] offer theoretical guarantees concerning the policy learning under perturbations within a certain range; (2) Other studies [17, 30, 31, 28] empirically examine the robustness of the training model to perturbations. 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 [17]. However, very few studies focus on the defenses against training-time attacks from a resilience perspective. In this work, we attempt to study a policy-resilience mechanism against environment-poisoning attacks on RL policy learning.

B Preliminaries

Hidden-Parameter Markov Decision Process. A Hidden Parameter MDP (HiP-MDP) [32] 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 [33]. 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 [32].

Federated Learning. The main idea of Federated Learning (FL) [34] 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 forbid to be collected/fused for processing model [35]. Recently, FL has addressed data isolation and privacy issues in RL-based applications, such as smart energy management [6], autonomous driving in Internet of Vehicles (IoV) [36] and optimal control of internet-of-things devices (IoT) [37].

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 [19]. 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".

C Methodology

C.1 Details of Preparation Stage

Figure 5 illustrates the learning of the server dynamics function based on environment information collected from all the client RL systems.

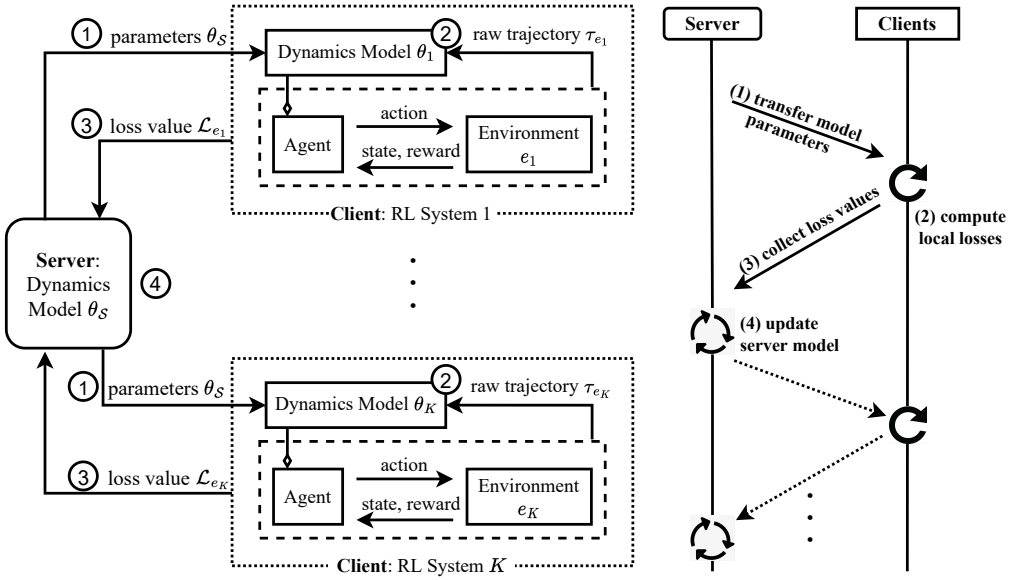


Figure 5: Illustration of preparation stage in policy-resilience mechanism.

The server dynamics model is learned through a combination of federated learning and meta-learning approaches [19], which is finalized as Algorithm 1.

Algorithm 1 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.[3] or Eq.[4]
    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 $\mathcal{S}$

Additionally, 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

$$\mathcal{L}(\mathcal{T}_e, \mathcal{D}_\theta) = \sum_{j=0}^{X} \|\mathcal{D}_\theta(s_j, a_j) - s_{j+1}\|_2^2, \tag{3}$$

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

$$\mathcal{L}(\mathcal{T}_e, \mathcal{D}_\theta) = -\mathbb{E}_{\langle s,a,s' \rangle \sim \mathcal{T}_e} \log \mathcal{D}_\theta(s'|s, a) \tag{4}$$

D Details in Recovery Stage

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 trajectory, therefore an understanding of the dynamics of the deployment environment (i.e., diagnosis) is crucial to finding an optimal policy. 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 [38, 39], 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 a 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) [20, 21] 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

$$V_h^{MVE}(r, s') = r + \left(\sum_{i=1}^{h} T^i \gamma^i R(s'_{i-1}, a'_{i-1}, s'_i)\right) + T^{h+1}\gamma^{h+1} Q^\pi_{\theta^-}(s'_h, a'_h), \tag{5}$$

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) [22]. 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

$$\sum_{t}^{t+H-1} \mathbb{E}_{s_{t+1}\sim\mathcal{D}_{\theta^*_{\hat{e}}}(s_t,a_t)}\big[r(s_t,a_t,s_{t+1})\big] \tag{6}$$

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 2 summarizes the operations performed during the diagnosis and recovery stages.

Algorithm 2 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 or 6, using trajectories imagined via $\mathcal{D}_{\theta'_{\hat{e}}}$

E Experiments

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.

E.1 Discrete State Domains

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

E.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).

Environment: The environment-poisoning attack is implemented on an RL agent performing navigation tasks in one 5×5 grid world as shown in Figure 6. The experiment environment (i.e., 3D grid world) is the same as that of TEPA, which simulates mountains or rugged terrain. In this 3D grid world, the agent's success of moving from one cell to the neighboring one is proportional to their relative elevation, changes in elevation will influence how the environment responds to the agent's action (i.e., environment dynamics). Therefore, the elevation setting is regarded as the hyper-parameters which can be tweaked by the attacker.

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 [34]. 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*.

E.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 hyper-parameters 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.

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 **??** 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 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.

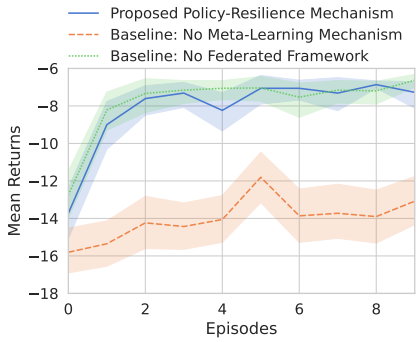


Figure 6: Performance comparison.

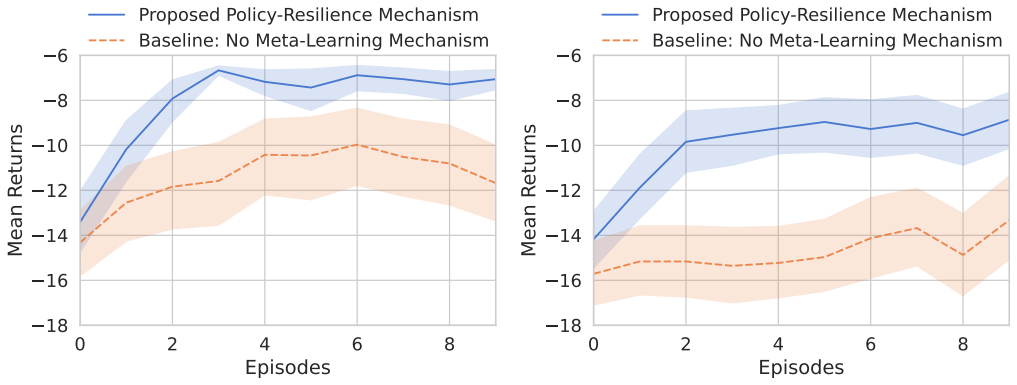
Actually, we have designed our policy-resilience mechanism in such a way that an agent in our RL will not need to invest additional resources or add-on capabilities for achieving resilience. Any agent may be provided with information on the environment and allowed to recover its poisoned policies accordingly, 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 **??** 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 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. 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 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 [34], 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 7: Comparison of policy resilience under various proportions of poisoned clients.

E.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.

E.2.1 Experiment Settings

Environments. The environment-poisoning attack, DBB-EPA, targets an RL agent performing a Cartpole task as depicted in Figure 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.

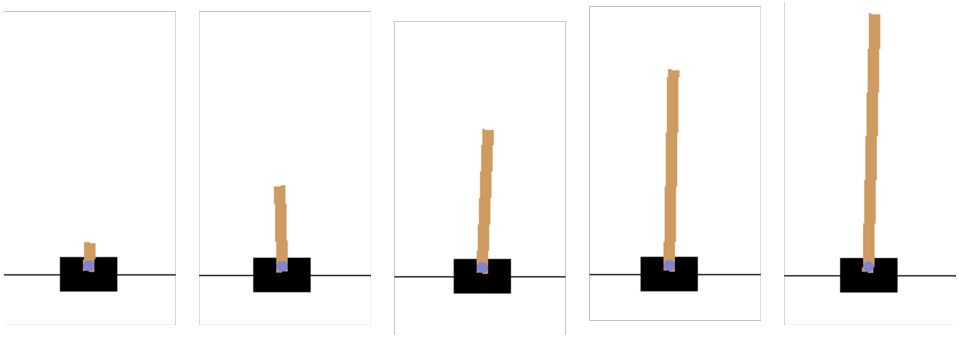


Figure 8: Illustration of Cartpole environment.

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 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*.

E.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 8). The agent learns the policy using model-free RL algorithms (e.g., DDPG) and recover it using MVE (see Chapter **??**) based on imagined trajectories generated by the learned dynamics model.

Figure 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.

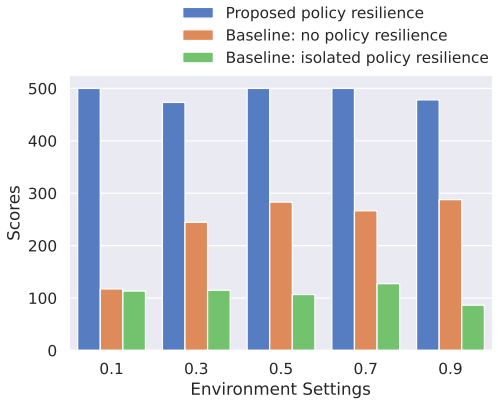


Figure 9: Comparison of policy-resilience performance on an *model-free* RL agent performing the Carpole task.

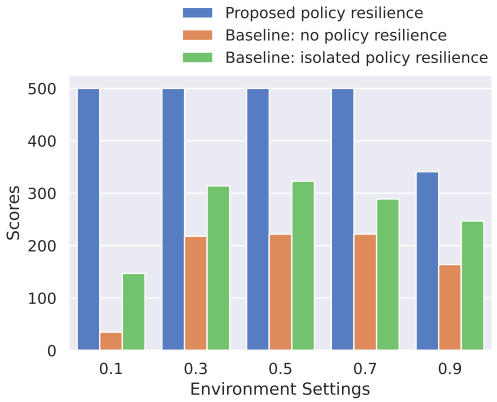


Figure 10: Comparison of policy-resilience performance on an *model-based* RL agent performing the Carpole task.

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 **??**). Except for this difference, all other settings are the same as those used with model-free RL agents. Figure 10

illustrates the performance of policy recovery in deployment environments where pole lengths are set at $0.1 \sim 0.9$ (see Figure 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.

We also observe a significant difference between *Baseline:isolated policy resilience* in Figures 10 and 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.

F Further 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 [40]. 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 [41], a state grid corporation and building grids [6], and a transport authority and autonomous vehicles [36]. Furthermore, as the Internet of Things (IoT) [42] and federated learning [34] 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 [43, 19]. 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 modeling, 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 [44–46] have been proposed to accurately represent the dynamics of the environment and these methods can be incorporated into our policy-resilience mechanisms.