Model and Method: Training-Time Attack for Cooperative Multi-Agent Reinforcement Learning

Siyang Wu ,*[1] **Tonghan Wang,***[1] **Xiaoran Wu**[2], **Jingfeng Zhang**[3],
Yujing Hu[4], **Changjie Fan**[4], **Chongjie Zhang**[1]
[1] Institute for Interdisciplinary Information Sciences, Tsinghua University, China
[2] Department of Computer Science and Technology, Tsinghua University, China
[3] RIKEN Center for Advanced Intelligence Project, Tokyo, Japan
[4] Fuxi AI Lab, NetEase, China
wu-sy19@mails.tsinghua.edu.cn
tonghanwang1996@gmail.com
wuxr17@tsinghua.org.cn
jingfeng.zhang@riken.jp
{huyujing, fanchangjie}@corp.netease.com
chongjie@tsinghua.edu.cn

Abstract

The robustness of deep cooperative multi-agent reinforcement learning (MARL) is of great concern and limits the application to real-world risk-sensitive tasks. Adversarial attack is a promising direction to study and improve the robustness of MARL but is largely under-studied. Previous work focuses on deploy-time attacks which may exaggerate attack performance because the MARL learner even does not anticipate the attacker. In this paper, we propose training-time attacks where the learner is allowed to observe and adapt to poisoned experience. For the stealthiness of attacks, we contaminate action sampling and restrict the attack budget so that non-adversarial agents cannot distinguish attacks from exploration noise. We derive two specific attack methods by modeling the influence of action-sampling on experience replay and further on team performance. Experiments show that our methods significantly undermine MARL algorithms by subtly disturbing the exploration-exploitation balance during the learning process.

1 Introduction

Deep multi-agent reinforcement learning (MARL) has achieved prominent progress in imitating many aspects of human cooperation such as policy decentralization [43, 38, 41, 29, 51], communication [20, 50], and organization [47, 48]. Despite these achievements, a common concern about existing cooperative MARL algorithms is their robustness. Recent research reports that they are sensitive to hyper-parameter settings [16], vulnerable to large variance [23], or have delicate convergence property [45]. The lack of robustness largely limits the application of MARL methods to risk-sensitive systems with a high demand on security.

Adversarial attack provides opportunities of evaluating, understanding, and further improving the robustness of MARL algorithms. Existing studies on this topic typically learn two policies simultaneously in the training stage: a *target policy* that solves the multi-agent task and an *adversarial policy*. The adversarial policy attacks the learned target policy at *test (or deployment) time* by choosing a victim agent and manipulating their action selection. Lin et al. [26] train the adversarial policy to

*Equal contribution

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

distinguish the action that reduces the team return most significantly, then the observation of the victim agent is contaminated after deploying so that it chooses this target action. Pham et al. [35] adopt a model-based approach and contaminate the observation to misguide the victim agent to select actions leading the team to pre-defined low-rewarding states.

However, in these *deploy-time attacks*, the target policy is trained without knowing that there is an adversarial attacker. In this situation, the unanticipated attacker may have exaggerated attack performance—an algorithm that is resilient to the attacks can also be dramatically undermined. For example, we find that agents become immune to deploy-time attacks when allowed to observe and learn from the poisoned experience. In Fig. 1, we show the test win rates of MARL agents (trained by QMIX [38]) on a map of the SMAC benchmark [39]. QMIX agents first learn for $1M$ timesteps without any attack. Then deploy-time attacks begin, and we find that QMIX performance drops significantly. We then allow the attacks to also appear at the training time so that agents observe the poisoned experience. In this case, we can see that the attack gradually loses its effectiveness.

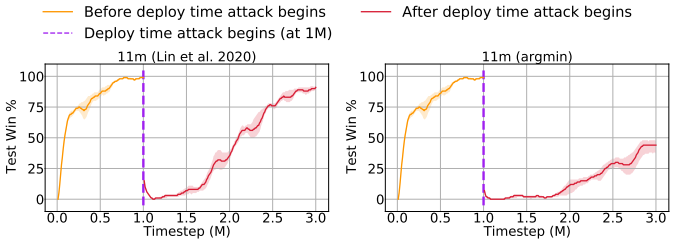


Figure 1: Deploy-time attacks cannot fully reveal an algorithms' robustness: if we allow adaption, the attacked algorithm quickly gets immune to the attack. Left: attack by Lin et al. [26]. Right: attack by choosing actions with the worst individual Q-value.

To more thoroughly reveal the algorithms' robustness, this paper proposes, to the best of our knowledge, the first *training-time attack* method for cooperative MARL agents. Our method corrupts the learning process of the target model and keeps silent at test or deployment time. The aim is to find attack methods that can effectively undermine the performance of the learned policies when the agents anticipate an adversarial attacker.

The training setting of cooperative MARL presents unique constraints on training-time attack model. Parameter sharing [38, 3, 25] is a widely used paradigm where agents share local information, parameters, and gradients. This sharing scheme means that we cannot inadvertently contaminate the observation, change the local policy parameters, or modify local gradients of the victim without being detected by other agents. For the stealthiness of the attack, we keep these possibly shared components intact but instead make use of the typically independent exploration noise (e.g., ϵ-greedy) and perturb the victim's action selection. The idea is to limit the attack so that normal agents cannot distinguish the attack from the exploration noise.

Given the limited attack budget, we derive two specific attack methods for perturbing the action selection. The derivation of both methods explicitly models the influence of the attacked action selection on the experience replay and further on the model updates. The first method derives the perturbations that increase the variance of model update gradients the most. The second method calculates the perturbations that reduce the team returns the most significantly after one model update.

We test our method on the SMAC [39] and MPE [29] benchmark and find it significantly undermines the learning performance on all the tested tasks. Visualizations of attacked gradients suggest that our attack methods may affect the learning performance by disturbing the exploration-exploitation balance. Moreover, we analyze the difference between the two attack methods and investigate the influence of the limitation of attack budgets on the performance of our attack methods.

2 Problem Formulation

We focus on fully cooperative multi-agent tasks that can be modelled as a **Dec-POMDP** [34] consisting of a tuple G=$\langle I, S, A, P, R, \Omega, O, n, \gamma\rangle$, where I is the finite set of n agents, $\gamma \in [0, 1)$ is the discount factor, and $s \in S$ is the true state of the environment. At each timestep, each agent i receives an observation $o_i \in \Omega$ drawn according to the observation function $O(s, i)$. We mainly study policy gradient methods. Each agent i has a policy π_{θ_i}, parameterized by θ_i, that recommends a probability distribution over the action space A given local action-observation history $\tau_i \in \mathrm{T} \equiv (\Omega \times A)^* \times \Omega$. The agent draws an action $a_i \in A$ from the distribution. Individual actions form a joint action $\boldsymbol{a} \in A^n$, which leads to a next state s' according to the transition function $P(s'|s, \boldsymbol{a})$, a reward $r = R(s, \boldsymbol{a})$ shared by all agents. Agents learn their polices to collectively

maximize the global team performance:

$$J(\theta) = \mathbb{E}_{s\sim P, \boldsymbol{a}\sim\boldsymbol{\pi}_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \boldsymbol{a}_t)|s_0], \tag{1}$$

where $\boldsymbol{\pi}_\theta$ is the joint policy and θ is its parameters. The joint action-value function is defined as $Q^{\boldsymbol{\pi}}_{tot}(s,\boldsymbol{a})$=$\mathbb{E}_{s_{0:\infty},\boldsymbol{a}_{0:\infty}}[\sum_{t=0}^{\infty}\gamma^t R(s_t,\boldsymbol{a}_t)|\ s_0$=$s, \boldsymbol{a}_0$=$\boldsymbol{a}, \boldsymbol{\pi}]$.

The **centralized training with decentralized execution** (CTDE) paradigm [8, 49] is popular for its ability to maintain decentralized execution while addressing learning non-stationarity. For policy-based learning, training a centralized critic with decentralized actors is an efficient approach that exploits the CTDE paradigm. MADDPG [29] learns deterministic policies updated by the gradients:

$$g = \mathbb{E}_{\boldsymbol{\tau},\boldsymbol{a}\sim\mathcal{D}}[\sum_i \nabla_{\theta_i}\pi_i(\tau_i)\nabla_{a_i} Q^{\boldsymbol{\pi}}_{tot}(s,\boldsymbol{a})|_{a_i=\pi_i(\tau_i)}],$$

where $\mathcal{D}$ is a replay buffer. COMA [9] updates stochastic policies using the gradients:

$$g = \mathbb{E}_{\boldsymbol{\pi}}[\sum_i \nabla_{\theta_i}\log\pi_i(a_i|\tau_i)A^{\boldsymbol{\pi}}_i(s,\boldsymbol{a})],$$

where $A^{\boldsymbol{\pi}}_i(s,\boldsymbol{a}) = Q^{\boldsymbol{\pi}}_{tot}(s,\boldsymbol{a}) - \sum_{a'_i} Q^{\boldsymbol{\pi}}_{tot}(s,(\boldsymbol{a}_{-i}, a'_i))$ is a counterfactual advantage ($\boldsymbol{a}_{-i}$ is the joint action other than agent i) that assigns credit and reduces gradient variance.

MADDPG and COMA use a joint critic, through which the suboptimality of one agent's policy can propagate and negatively affect policy learning of other agents. DOP [51] solves this *centralized-decentralized mismatch* issue by learning a centralized but factored critic:

$$Q^{\boldsymbol{\pi}}_{tot}(s,\mathbf{a};\phi) = \textstyle\sum_i k_i(\boldsymbol{\tau})Q^{\boldsymbol{\pi}}_i(\boldsymbol{\tau},a_i;\phi_i) + b(\boldsymbol{\tau}),$$

where ϕ is the parameters of the critic and $\boldsymbol{\tau}$ is the joint observation. DOP also uses this decomposed critic to enable off-policy learning for stochastic policies and credit assignment for deterministic policies.

3 Attack model and methods

In this section, we introduce our TRaining-time Attack Model and methods (TRAM) for cooperative multi-agent reinforcement learning. Similar to previous work on deploy-time attack, we allow attacks for one of the cooperative agents and expect that the attack is stealthy. Without loss of generality, we suppose that, among agents $1, 2, \ldots, n$, the k-th one is the poisoned.

3.1 Attack model

Training-time attacks disturb the learning process of an MARL policy and do not interfere after the policy is deployed or is being tested. Since in many cooperative MARL algorithms agents share their parameters and local information, we do not contaminate local parameters, observations, and gradients. Rather, we adopt a sampling-based attack model: given the output of the local policy π_k, we perturb the action selection and limit the perturbations so that other agents would confuse the attack with the exploration noise.

Formally, we denote the perturbed action distribution of agent k by π'_k. The global policy when agent k is adversarial is:

$$\boldsymbol{\pi}'(\boldsymbol{a}|\boldsymbol{\tau}) = \pi'_k(a_k|\tau_k)\Pi_{i\neq k}\pi_i(a_i|\tau_i). \tag{2}$$

To limit attacks, we (1) restrict the attack frequency. Each timestep, the poisoned agent has a probability f_A to be adversarial (samples its action from π'_k), and in other situations, it is non-adversarial (uses π_k). Moreover, we (2) limit the KL divergence between π'_k and π_k: $D_{\mathrm{KL}}(\pi'_k\|\pi_k) = \sum_a \pi'_k(a|\tau_k)\log\frac{\pi'_k(a|\tau_k)}{\pi_k(a|\tau_k)}$ within a threshold, KL^U. In this way, from the perspective of other agents, the log likelihood that an agent is abnormal due to attack:

$$\begin{aligned}\mathcal{L} =& f_A\sum_a \pi'_k(a|\tau_k)\log\pi_k(a|\tau_k) + (1-f_A)\sum_a \pi_k(a|\tau_k)\log\pi_k(a|\tau_k)\\ =& -f_A H(\pi'_k,\pi_k) - (1-f_A)H(\pi_k) = -f_A H(\pi'_k) - f_A D_{\mathrm{KL}}(\pi'_k\|\pi_k) - (1-f_A)H(\pi_k),\end{aligned} \tag{3}$$

where $H(\cdot,\cdot)$ and $H(\cdot)$ stands for cross entropy and entropy, is also greater than a pre-defined threshold. The question is how to obtain π'_k that effectively undermines the team cooperation performance under the restricted attack budget.

Suppose that π_k is the softmax of the logits l_k. The attacker computes a perturbation vector δ of the same dimension as l_k and adds it to l_k. Let $l'_k = \delta + l_k$. Then π'_k is calculated as $\mathrm{softmax}(l'_k)$. In this paper, we propose two methods to derive δ. In Sec. 3.2, we introduce a method that tries to increase the variance of gradients for model updates. In Sec. 3.3, we introduce a method that tries to decrease the expected return after one model update. In both methods, we write $\delta = \beta \cdot \hat{\delta}$ where β is a scaling factor controlling the attack strength and $\hat{\delta}$ is a unit vector in the direction of the adversarial attack. Our methods first compute $\hat{\delta}$. Then we select a β so that the log likelihood (Eq. 3) does not exceed the threshold (discussed in detail in Appendix B). In the derivation of our methods, we use $\rho(s, \boldsymbol{a}) = E_{\tilde{\boldsymbol{\pi}}}[\sum_{t=0}^{T} \frac{\mathbf{1}_{s_t=s, \boldsymbol{a}_t=\boldsymbol{a}}}{T}]$, where T is the episode horizon, to denote the distribution of the state-action pair (s, a) during experience collection using the attacked policy $\tilde{\boldsymbol{\pi}} = f_A \boldsymbol{\pi}' + (1 - f_A)\boldsymbol{\pi}$.

3.2 Variance-based attack

The idea of the first attack method is to increase the variance of policy gradients. Intuitively, gradients with larger variance will render the learning process unstable.

Given the buffer with the sample density ρ and the local policy parameters θ^t, the gradient for updating the policy is $\sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A(s, \boldsymbol{a}) (\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau}))^\top$. This gradient can be regarded as a multivariate random variable. The covariance matrix of this multivariate random variable is:

$$\begin{aligned}\Sigma = &\sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A^2(s, \boldsymbol{a}) \left(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right)^\top \left(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta}(\boldsymbol{a}|\boldsymbol{\tau})\right) \\ &- \left(\sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A(s, \boldsymbol{a}) \nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right)^\top \cdot \left(\sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A(s, \boldsymbol{a}) \nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right)\end{aligned} \tag{4}$$

For increasing the gradient variance, we can maximize the trace of Σ:

$$\mathrm{tr}(\Sigma) = \sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A^2(s, \boldsymbol{a}) \left\|\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right\|^2 - \left\|\sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A(s, \boldsymbol{a}) \nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right\|^2 .$$

The second term $\| \sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A(s, \boldsymbol{a}) \nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\|^2$ is the norm of gradients for θ. In most widely used neural network optimization methods like RMSProp, Adam, etc., the gradients are normalized. Since we aim to maximize $\mathrm{tr}(\Sigma)$, this second term can be ignored, and the objective we propose to maximize becomes the trace of the *auto-correlation* matrix of θ's gradient:

$$R = \sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A^2(s, \boldsymbol{a}) \left(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right)^\top \left(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta}(\boldsymbol{a}|\boldsymbol{\tau})\right), \tag{5}$$

$$\mathrm{tr}(R) = \sum_{s,\boldsymbol{a}} \rho(s, \boldsymbol{a}) A^2(s, \boldsymbol{a}) \left\|\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right\|^2 . \tag{6}$$

To maximize $\mathrm{tr}(R)$, we calculate its gradients with respect to the adversarial vector δ at the point of the intact policy, *i.e.*, $\hat{\delta} = \texttt{Normalize}(\nabla_\delta \mathrm{tr}(R)|_{\delta=0})$. Because that δ influences ρ, and further influences R, $\nabla_\delta \mathrm{tr}(R)$ can be computed as:

$$\nabla_\delta \mathrm{tr}(R) = \sum_{s,\boldsymbol{a}} \nabla_\delta \rho(s, \boldsymbol{a}) A^2(s, \boldsymbol{a}) \left\|\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})\right\|^2 . \tag{7}$$

We expand $\nabla_\delta \rho(s, \boldsymbol{a})$ by the chain rule:

$$\nabla_\delta \rho(s, \boldsymbol{a}) = \nabla_{\tilde{\boldsymbol{\pi}}} \rho(s, \boldsymbol{a}) \nabla_\delta \tilde{\boldsymbol{\pi}}. \tag{8}$$

It follows that $\tilde{\boldsymbol{\pi}}(\boldsymbol{a}|\boldsymbol{\tau}) = \frac{\rho(s,\boldsymbol{a})}{\sum_{\boldsymbol{a}'} \rho(s,\boldsymbol{a}')}$. Since the gradient $\nabla_\delta \mathrm{tr}(R)$ will be normalized when calculating $\hat{\delta}$ and $\tilde{\boldsymbol{\pi}}(\boldsymbol{a}|\boldsymbol{\tau}) \propto \rho(s, \boldsymbol{a})$, we can approximate the first term in Eq. 8 as 1.And for $\nabla_\delta \tilde{\boldsymbol{\pi}}$,

$$\nabla_\delta \tilde{\boldsymbol{\pi}}(\boldsymbol{a}|\boldsymbol{\tau}) = \frac{\partial \tilde{\boldsymbol{\pi}}(\boldsymbol{a}|\boldsymbol{\tau})}{\partial \boldsymbol{\pi}'(\boldsymbol{a}|\boldsymbol{\tau})} \nabla_\delta \boldsymbol{\pi}'(\boldsymbol{a}|\boldsymbol{\tau}) = f_A \nabla_\delta (\pi'_k(a_k|\tau_k)) \Pi_{i \neq k} \pi_i(a_i|\tau_i). \tag{9}$$

And

$$\frac{\partial \pi'_k(a|\tau_k)}{\partial \delta(a')} = \mathbf{1}_{a=a'}\pi'_k(a|\tau_k) - \pi'_k(a|\tau_k)\pi'_k(a'|\tau_k). \tag{10}$$

where $\delta(a')$ is the a'-th element in δ. (Recall that δ is a vector of the same dimension as π_k.)

By incorporating Eq. 8, 9, and 10 into Eq. 7, we get the expression for calculating variance-based adversarial vectors.

3.3 J-based attack

For the second method, we want the attack to minimize the team performance J (Eq. 1) after one model update. Specifically, given the local policy parameters at timestep t, θ^t, we derive the adversarial gradient to minimize $J(\theta^{t+1})$. Since the attack gradient δ influences the empirical sample distribution ρ, which further influences θ^{t+1}, we write θ^{t+1} as a function of δ, i.e. $\theta^{t+1}(\delta)$.

To minimize $J(\theta^{t+1}(\delta))$, we perform one step of gradient descent from the intact policy, *i.e.*, $\hat{\delta} = \texttt{Normalize}(-\nabla_\delta J(\theta^{t+1}(\delta_0))|_{\delta_0=0})$. The question then is how to calculate $\nabla_\delta J(\theta^{t+1}(\delta))$.

By the Taylor expansion of $J(\theta^{t+1}(\delta))$, we have:

$$J(\theta^{t+1}(\delta)) \approx J(\theta^t) + \nabla_{\theta^t} J(\theta^t)(\theta^{t+1}(\delta) - \theta^t). \tag{11}$$

Assuming that the policy and parameters do not change much in one model update, we can ignore the higher-order terms in the Taylor expansion. Therefore

$$\nabla_\delta J(\theta^{t+1}(\delta)) \approx \nabla_\delta [J(\theta^t) + \nabla_{\theta^t} J(\theta^t)(\theta^{t+1}(\delta) - \theta^t)] = \nabla_{\theta^t} J(\theta^t)\nabla_\delta(\theta^{t+1}(\delta) - \theta^t). \tag{12}$$

Since

$$\theta^{t+1} = \theta^t + \alpha_\pi \sum_{s,\boldsymbol{a}} \rho(s,\boldsymbol{a}) A(s,\boldsymbol{a})(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau}))^\top,$$

where α_π is the learning rate for policies and $A(s,\boldsymbol{a})$ is the advantage function, we have

$$\begin{aligned} \nabla_\delta J(\theta^{t+1}(\delta)) &\approx \nabla_{\theta^t} J(\theta^t)\nabla_\delta [\alpha_\pi \sum_{s,\boldsymbol{a}} \rho(s,\boldsymbol{a}) A(s,\boldsymbol{a})(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau}))^\top], \\ &= \nabla_{\theta^t} J(\theta^t)[\alpha_\pi \sum_{s,\boldsymbol{a}} A(s,\boldsymbol{a})(\nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau}))^\top (\nabla_\delta \rho(s,\boldsymbol{a}))]. \end{aligned} \tag{13}$$

The first term of Eq. 13 ($\nabla_{\theta^t} J(\theta^t)$) is the classic policy gradients. However, it can not be obtained when calculating δ because δ is computed before sampling. Therefore, we use the policy gradient at the previous timestep to approximate this term. This approximation is reasonable when the policy does not change much in one model update.

The second term of Eq. 13 can be calculated in a similar way to the case of variance-based attack. By incorporating Eq. 8, 9, and 10 into Eq. 13, we get the adversarial vector δ for J-based attack

4 Experiments

In this section, we design experiments to answer the following questions. (1) How do J-based and variance-based attacks influence the team cooperation performance? How do these two attack methods compare to each other? (Sec. 4.1) (2) How do the attack frequency and KL constraints influence the attack? (Sec. 4.2) We also show visualizations of the attack gradients to shed light on how our methods work.

We use DOP as the target MARL algorithm in this section and leave the experiments of attacking other MARL algorithms in Appendix D. We evaluate TRAM on the benchmark of SMAC[2] [39] and MPE [29]. All hyperparameters for training DOP agents are the same on all maps and are consistent with the default setting recommended by the authors [51]. Hyperparameter settings of our methods

[2]Our experiments are based on the PyMARL framework, which uses SC2.4.6.2.6923. Note that performance is not always comparable among versions.

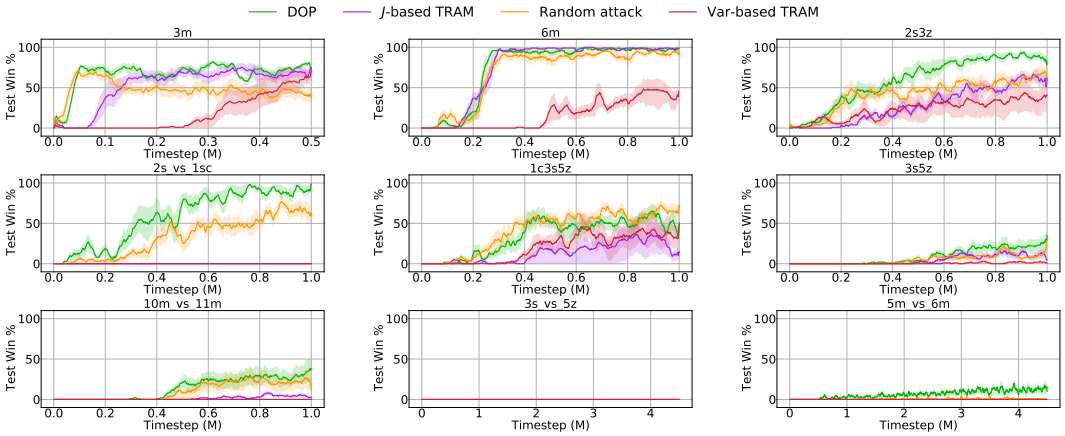


Figure 2: Performance of our methods compared against a random attack baseline on SMAC maps.

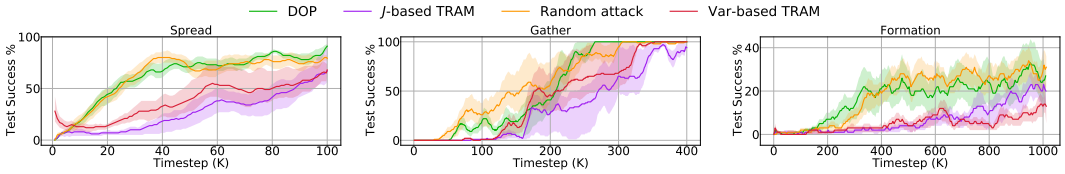


Figure 3: Performance of our methods compared against a random attack baseline on MPE tasks.

can be found in Appendix A. To ensure fair evaluation, we carry out all our experiments with five random seeds, and the average performance is shown with a 95% confidence interval.

We use three fully cooperative tasks for the MPE benchmark. In `Spread`, three agents are expected to cover three randomly initialized landmarks. For `Gather`, three agents will be collectively rewarded if they arrive at the single randomly initialized landmark simultaneously. In `Formation`, a team reward is given to the team if the four members form a square centered at a random landmark.

4.1 Attack performance on SMAC and MPE

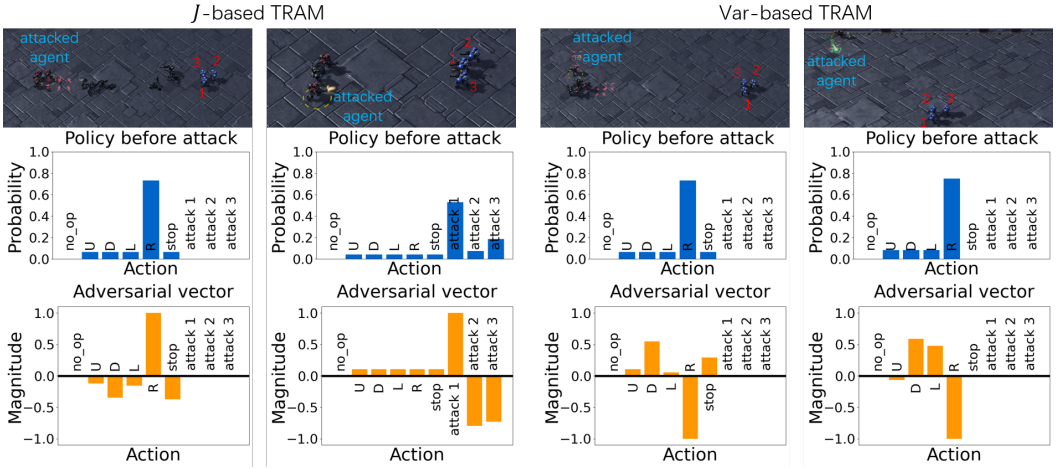
We compare the two proposed attack methods against a training-time attack baseline that randomly generates the attack gradient δ. For our methods and the baseline algorithm, we set the attack frequency f_A to 1 and KL^U to 6. For all maps, all adversarial methods attack one agent. We attack a Zealot in the map `2s3z` and `3s5z` and attack the Colossus in map `1c3s5z`.

From the results shown in Fig. 2 and Fig. 3, we can see that the proposed methods can significantly undermine the learning of DOP agents, while the random attack baseline has a much weaker attack performance. For example, on the map `2s_vs_1sc` and `10m_vs_11m`, both DOP and the random attack baseline attack can learn a satisfactory policy for solving the task, but when attacked by TRAM, the win rates drop to 0. Such a performance gap can be observed consistently across the SMAC and MPE benchmark, highlighting the effectiveness of our method. Generally, variance-based TRAM is more effective than J-based TRAM. For example, on `3m` and `6m`, variance-based attack is very effective while J-based attack only slightly influences the performance of DOP.

Why is our method more effective? To understand the observed experimental results, we compare the log likelihood of being attacked when the attack frequency (f_A=1) and KL upper (KL^U=6) bound are fixed. Results are shown in Table 1. In this table, all values are greater than the pre-defined likelihood threshold, and a smaller value means more salient attack behaviors. We can see that variance-based attack has the smallest values among all the tested methods while the random attack has the largest. The reason is that the adversarial gradients' direction selected by the random attack method is not effective. Even when there are no limits on the attack frequency and the KL bound, the attack behaviors still can not be significant enough. By contrast, variance-based TRAM selects a better attack direction and thus can make a better use of the limited attack budget, which explains its superior performance.

Table 1: Variance-based TRAM makes a better use of the limited attack budget: under a fixed attack frequency and a fixed KL bound, the log likelihood of being attacked is shown. With all log likelihoods within the detectable threshold, var-based TRAM has the most salient attack behavior.

	3m	6m	2s3z	2s_vs_1sc	1c3s5z	3s5z	10m_vs_11m
Var-based TRAM	-3.87	-4.32	-3.41	-3.36	-3.30	-3.08	-4.25
J-based TRAM	-2.43	-2.88	-1.65	-3.08	-1.35	-1.26	-2.89
Random attack	-1.53	-2.28	-1.76	-2.45	-0.75	-1.26	-1.53



(a) J-based TRAM tends to enlarge the the probability of the most likely action.

(b) Variance-based TRAM tends to reduce the probability of the most likely action.

Figure 4: Examples of TRAM attack on 3m. We show game screenshots, the policy before attacks, and the normalized adversarial vector $\hat{\delta}$ computed by TRAM.

How does our method interfere with the learning process? To understand how the proposed attack methods work, we present the replays of our methods on the SMAC map 3m in Fig. 4 and investigate how the attacked agent's policy is affected.

J-based attack. Fig. 4(a) shows two frames from two different trajectories during learning when one Marine is attacked by J-based TRAM. We show the policy before the attack and the attack gradient $\hat{\delta}$. From these results, we can see that, for this map, the probability of the most likely action is further enlarged. In this way, J-based TRAM prevents exploration and thereby hurts the performance.

Variance-based attack. Fig. 4(b) shows two frames from two different trajectories during learning when one Marine is attacked by variance-based TRAM. Again, we show the intact policy and the adversarial gradient $\hat{\delta}$ computed by the variance-based method. From these results, we observe that the probability of the most likely action is dramatically reduced. In this way, variance-based TRAM makes the attacked agent exploit less and explore more. The drawback is that it will be harder to make full trajectories of good quality for training.

4.2 Influence of attack budgets

In this sections, we carry out experiments on SMAC maps to study the influence of the limited attack budget on the performance of out method. Specifically, we compare the performance under different attack frequencies and KL-divergence upper bound.

We first fix the KL upper bound KL^U=6 and test with attack frequencies 0.4, 0.6, 0.8, and 1.0 in Fig. 5. It can be observed that better attack performance can be obtained with a higher attack frequency for all attack methods. Random attacks are less affected by the frequency, while the performance of variance-based TRAM varies the most with the frequency. We then fix the attack frequency f_A=1 and test with KL divergence thresholds 2, 4, and 6. Results are shown in Fig. 6. Again, we observe that a loose budget leads to better attack performance. Compared to attack frequency, KL upper bounds have a more significant influence on the performance. Moreover, J-based TRAM is less affected

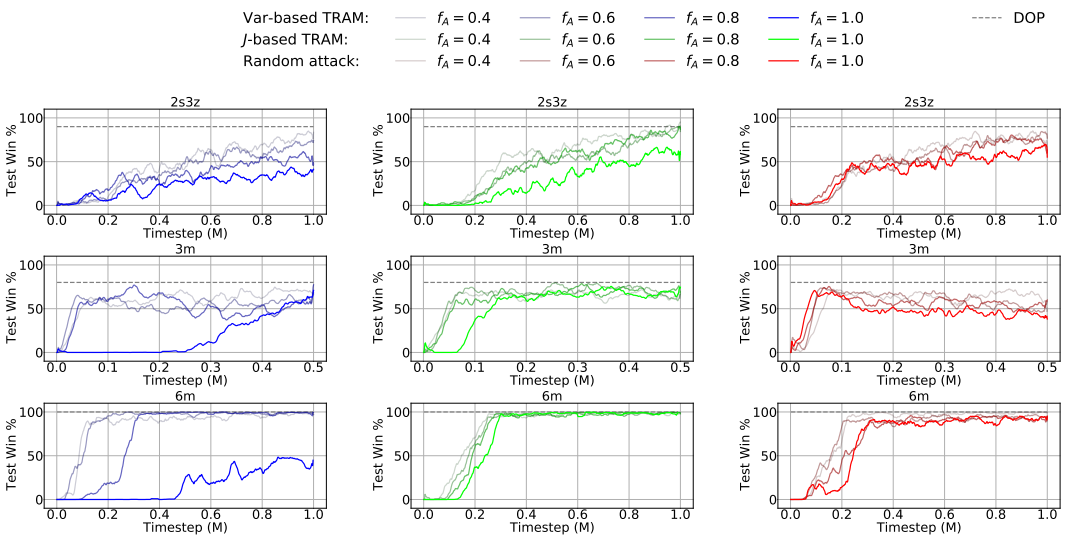


Figure 5: Performance of our methods and the random attack baseline under different attack frequencies. For these experiments, $KL^U = 6$. Because curves are distinguished by transparencies, we ignore the confidence intervals in this figure. We show them in Fig. 11.

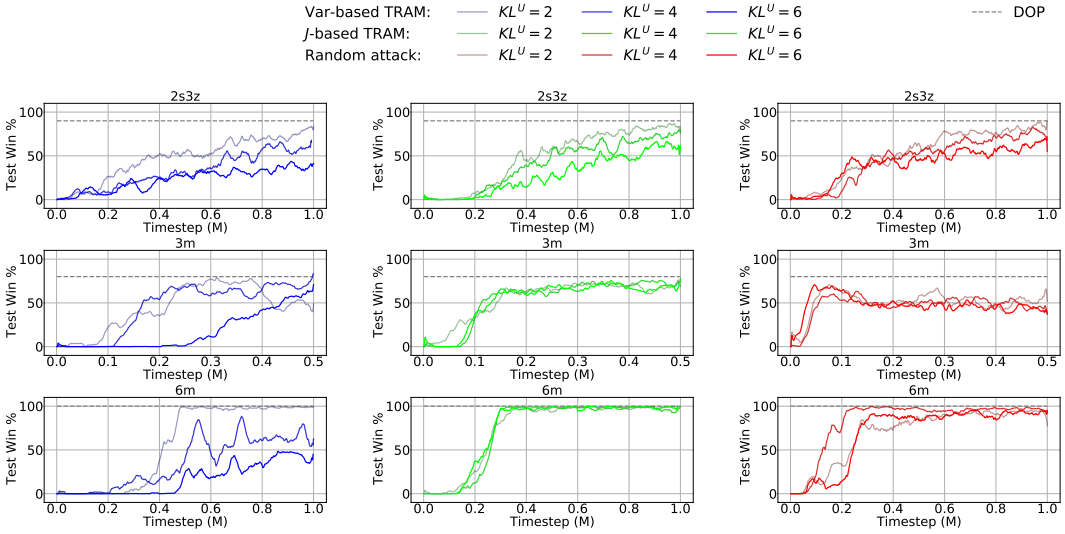


Figure 6: Performance of our methods and the random attack baseline under different KL-divergence upper bounds. For these experiments, $f_A = 1$. Because curves are distinguished by transparencies, we ignore the confidence interval here. We display them in Fig. 12.

compared to variance-based TRAM. These results again indicate that the attack direction selected by variance-based TRAM makes a better use of the attack budget.

5 Related Work

In this section, we introduce attack methods for deep learning, reinforcement learning, and multi-agent reinforcement learning that are related to our work.

Attacks for deep learning The attacks in deep neural networks occur at inference or training stages. Inference-stage attacks include adversarial attack [44, 32], model inversion attack [10], membership inference attack [40], model extraction attacks [19], adversarial reprogramming [7], etc. The training-stage attacks include poisoning training data [31] through injecting visible backdoors [12] or invisible backdoors [4], modifying networks [6, 13], etc. In this work, we focus on the training-time attack counterpart in the domain of cooperative multi-agent reinforcement learning.

Attacks for reinforcement learning Reinforcement learning (RL) is vulnerable to adversarial attacks, where an imperceptible perturbation is added to one of the RL's components (such as states, actions, rewards, or models) to cause malfunctions. The adversary can attack the state observations through perturbing the sensors or environments [2, 18, 27, 22, 5, 30, 37]. The adversary can also control the actuators to perturb the actions [24, 28] or even flip the rewards [15]. Furthermore, the adversary can poison the model parameters [21, 17, 1]. There is also a generic attack framework [42]. Our method perturbs the logits of policy and differs by working under a cooperative MARL setting.

Attacks for competitive two-agent reinforcement learning Previous works under this setting [11, 14, 46] first train a victim agent and then fix the victim and train an adversarial agent to undermine the victim's performance by changing the adversarial agent's behavior. The attack is performed by training the adversarial agent to maximize its reward in a zero-sum game[11], both maximize its reward and minimize the victim agent's reward for non zero-sum game[14], or learn several trigger actions to exploit the weakness of the victim agent[46]. These works differ from ours because (1) we consider a cooperative MARL setting and the number of agents may be larger than two; and (2) we train the attack policy and other agents' policy simultaneously.

Attacks for cooperative multi-agent reinforcement learning Previous attack methods in this area mainly focus on the centralized training with decentralized execution (CTDE) scheme where agents can access global information during training and make individual decisions based on local information when executing. Prior works study several aspects of the deploy-time attack. For example, Lin *et al*. [26] and Pham *et al*. [35] study how to attack an agent's observations at deployment time to reduce the team's return. Nisioti *et al*. [33] consider the adversary that randomly selects attacking timesteps in the deployment time and performs adversarial selections of some agents to manipulate their actions. Phan *et al*. [36] consider unexpected failures of some agents at deployment time. To the best of our knowledge, we are the first to present a training-time attack to complement the study of attacking methods for the robustness evaluation of cooperative MARL.

6 Conclusion

In this paper, we propose training-time attacks and expect that they can provide a new perspective of studying the robustness of cooperative MARL algorithms. We propose a model for training-time MARL attacks where the possibly shared components, such as parameters, gradients, and observations, remain intact, but the action selection is attacked under the cover of exploration noise. To effectively utilize the restricted attack budget, we derive two attack methods that decreases the expected team return and increases the gradient variance, respectively. While we do not see obvious negative societal impacts of our method, for future works, we plan to investigate attacks for value-based cooperative MARL methods and for the critic learning in the policy-based learning domain. It is also important to develop novel, secure MARL defense schemes.

References

[1] Vahid Behzadan and William H. Hsu. Adversarial exploitation of policy imitation. In *Proceedings of the Workshop on Artificial Intelligence Safety 2019 co-located with the 28th International Joint Conference on Artificial Intelligence, AISafety@IJCAI*, volume 2419 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[2] Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition - 13th International Conference, MLDM*, volume 10358 of *Lecture Notes in Computer Science*, pages 262–275. Springer, 2017.

[3] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[4] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[5] George W. Clark, Michael V. Doran, and William Glisson. A malicious attack on the machine learning policy of a robotic system. In *17th IEEE International Conference On Trust, Security*

And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, pages 516–521. IEEE, 2018.

[6] Jacob Dumford and Walter J. Scheirer. Backdooring convolutional neural networks via targeted weight perturbations. In *2020 IEEE International Joint Conference on Biometrics, IJCB*, pages 1–9. IEEE, 2020.

[7] Gamaleldin F. Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial reprogramming of neural networks. In *International Conference on Learning Representations*, 2019.

[8] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[9] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

[11] Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. *CoRR*, abs/1905.10615, 2019.

[12] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[13] Chuan Guo, Ruihan Wu, and Kilian Q Weinberger. Trojannet: Embedding hidden trojan horse models in neural networks. *arXiv preprint arXiv:2002.10078*, 2020.

[14] Wenbo Guo, Xian Wu, Sui Huang, and Xinyu Xing. Adversarial policy learning in two-player competitive games. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3910–3919. PMLR, 18–24 Jul 2021.

[15] Yi Han, Benjamin I. P. Rubinstein, Tamas Abraham, Tansu Alpcan, Olivier Y. de Vel, Sarah M. Erfani, David Hubczenko, Christopher Leckie, and Paul Montague. Reinforcement learning for autonomous defence in software-defined networking. In *Decision and Game Theory for Security - 9th International Conference, GameSec 2018, Seattle, WA, USA, October 29-31, 2018, Proceedings*, volume 11199 of *Lecture Notes in Computer Science*, pages 145–165. Springer, 2018.

[16] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and SW Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*, 2021.

[17] Mengdi Huai, Jianhui Sun, Renqin Cai, Liuyi Yao, and Aidong Zhang. Malicious attacks against deep reinforcement learning interpretations. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 472–482. ACM, 2020.

[18] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *5th International Conference on Learning Representations, ICLR, Workshop Track Proceedings*. OpenReview.net, 2017.

[19] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium, USENIX Security*, pages 1345–1362. USENIX Association, 2020.

[20] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, pages 7254–7264, 2018.

[21] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. Trojdrl: Trojan attacks on deep reinforcement learning agents. *arXiv preprint arXiv:1903.06638*, 2019.

[22] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[23] Jakub Kuba, Muning Wen, Linghui Meng, Haifeng Zhang, David Mguni, Jun Wang, Yaodong Yang, et al. Settling the variance of multi-agent policy gradients. *Advances in Neural Information Processing Systems*, 34, 2021.

[24] Xian Yeow Lee, Sambit Ghadai, Kai Liang Tan, Chinmay Hegde, and Soumik Sarkar. Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 4577–4584. AAAI Press, 2020.

[25] Chenghao Li, Chengjie Wu, Tonghan Wang, Jun Yang, Qianchuan Zhao, and Chongjie Zhang. Celebrating diversity in shared multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 35, 2022.

[26] Jieyu Lin, Kristina Dzeparoska, Sai Qian Zhang, Alberto Leon-Garcia, and Nicolas Papernot. On the robustness of cooperative multi-agent reinforcement learning. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 62–68. IEEE Computer Society, 2020.

[27] 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 Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 3756–3762. ijcai.org, 2017.

[28] Guanlin Liu and Lifeng Lai. Provably efficient black-box action poisoning attacks against reinforcement learning. *CoRR*, abs/2110.04471, 2021.

[29] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[30] Yuzhe Ma, Xuezhou Zhang, Wen Sun, and Jerry Zhu. Policy poisoning in batch reinforcement learning and control. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[31] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[32] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, pages 427–436. IEEE Computer Society, 2015.

[33] Eleni Nisioti, Daan Bloembergen, and Michael Kaisers. Robust multi-agent q-learning in cooperative games with adversaries. 2021.

[34] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[35] Nhan Pham, Lam M. Nguyen, Jie Chen, Thanh Lam Hoang, Subhro Das, and Tsui-Wei Weng. Evaluating robustness of cooperative MARL, 2022.

[36] Thomy Phan, Lenz Belzner, Thomas Gabor, Andreas Sedlmeier, Fabian Ritz, and Claudia Linnhoff-Popien. Resilient multi-agent reinforcement learning with adversarial value decomposition. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pages 11308–11316. AAAI Press, 2021.

[37] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7974–7984. PMLR, 13–18 Jul 2020.

[38] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4292–4301, 2018.

[39] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[40] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP*, pages 3–18. IEEE Computer Society, 2017.

[41] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896, 2019.

[42] Yanchao Sun and Furong Huang. Vulnerability-aware poisoning mechanism for online RL with unknown dynamics. *CoRR*, abs/2009.00774, 2020.

[43] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[44] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[45] Jianhao Wang, Zhizhou Ren, Beining Han, Jianing Ye, and Chongjie Zhang. Towards understanding cooperative multi-agent q-learning with value factorization. *Advances in Neural Information Processing Systems*, 34, 2021.

[46] Lun Wang, Zaynah Javed, Xian Wu, Wenbo Guo, Xinyu Xing, and Dawn Song. BACKDOORL: backdoor attack against competitive reinforcement learning. *CoRR*, abs/2105.00579, 2021.

[47] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[48] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[49] Tonghan Wang, Jianhao Wang, Wu Yi, and Chongjie Zhang. Influence-based multi-agent exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[50] Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. Learning nearly decomposable value functions with communication minimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[51] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Dop: Off-policy multi-agent decomposed policy gradients. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

A Hyperparameters and Infrastructures

For all experiments on the SMAC benchmark, we use the default reward and observation settings. For our method, we set the discount factor γ to 0.99 for all experiments. We use RMSprop with $\alpha = 0.99$ and no momentum or weight decay for the optimization of both the critic and actors. The learning rate for the critic is 0.0001 and the learning rate for actors is 0.0005. The critic and actors have the same network architecture as DOP [51]. 16 episodes are sampled from the on-policy buffer each time to train both the critic and actors. The on-policy buffer has a buffer size of 32. We run 4 parallel environments to collect data. ϵ-greedy is used during exploration. We let ϵ first anneal linearly from 1.0 to 0.05 over $500k$ time steps and then keep constant for the rest of the training. All experiments are conducted on NVIDIA GEFORCE RTX 3090 GPUs and Intel Xeon Gold 6248R CPUs. We use 1 GPU for each experiment.

B Choice of β

When computing $\hat{\delta}$, we select a β such that $D_{\mathrm{KL}}(\pi'_k \| \pi_k) \leq KL^U$. In this way, $\mathcal{L} = -f_A H(\pi'_k) - f_A D_{\mathrm{KL}}(\pi'_k \| \pi_k) - (1 - f_A)H(\pi_k) \geq -f_A H(\pi'_k) - f_A KL^U - (1 - f_A)H(\pi_k)$ can be bounded. There can be multiple values of β such that $D_{\mathrm{KL}}(\pi'_k \| \pi_k) \leq KL^U$ is satisfied. We set β to the largest one. To find such a β, we first prove in Lemma. B.1 that $D_{\mathrm{KL}}(\pi'_k \| \pi_k)$ monotonically increases as β increases.

Lemma B.1. *Let N be the number of actions, i.e. $N = |A|$. Suppose the policy π is the softmax of the logits l, and $\hat{\delta}$ is a vector of the same dimension as l. Let $\pi'(\beta) = \mathrm{softmax}(l + \beta\hat{\delta})$. Then for $\beta \geq 0$, $\frac{dD_{\mathrm{KL}}(\pi'(\beta)\|\pi)}{d\beta} \geq 0$.*

Proof. Let $Z = \sum_{a=1}^{N} \exp(l(a))$, $Z'(\beta) = \sum_{a=1}^{N} \exp(l(a) + \beta\hat{\delta}(a))$, then

$$\frac{dD_{\mathrm{KL}}(\pi'(\beta)\|\pi)}{d\beta} = \frac{d}{d\beta}[\sum_{a=1}^{N} \frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)} \log \frac{\exp(l(a) + \beta\hat{\delta}(a))/Z'(\beta)}{\exp(l(a))/Z}] \tag{14}$$

$$= \sum_{a=1}^{N} \frac{d}{d\beta}(\frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)}) \log \frac{\exp(l(a) + \beta\hat{\delta}(a))/Z'(\beta)}{\exp(l(a))/Z} + \sum_{a=1}^{N} \frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)} \frac{d}{d\beta}(\log \frac{\exp(l(a) + \beta\hat{\delta}(a))/Z'(\beta)}{\exp(l(a))/Z}). \tag{15}$$

Because

$$\frac{d}{d\beta}(\frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)}) = \frac{\exp(l(a) + \beta\hat{\delta}(a))\hat{\delta}(a)}{Z'(\beta)} - \frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)^2} \frac{dZ'(\beta)}{d\beta} \tag{16}$$

and

$$\frac{d}{d\beta}(\log \frac{\exp(l(a) + \beta\hat{\delta}(a))/Z'(\beta)}{\exp(l(a))/Z}) = \frac{d}{d\beta} \log(\frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)}) \tag{17}$$

$$= \frac{\frac{d}{d\beta} \exp(l(a) + \beta\hat{\delta}(a))}{\exp(l(a) + \beta\hat{\delta}(a))} - \frac{\frac{d}{d\beta} Z'(\beta)}{Z'(\beta)} \tag{18}$$

$$= \hat{\delta}(a) - \frac{\frac{d}{d\beta} Z'(\beta)}{Z'(\beta)}, \tag{19}$$

by incorporating Eq. 16 and 19 into Eq. 15, we have

$$\frac{dD_{\mathrm{KL}}(\pi'(\beta)\|\pi)}{d\beta} \tag{20}$$

$$= \sum_{a=1}^{N} [\frac{\exp(l(a) + \beta\hat{\delta}(a))\hat{\delta}(a)}{Z'(\beta)} - \frac{\exp(l(a) + \beta\hat{\delta}(a))}{Z'(\beta)^2} \frac{dZ'(\beta)}{d\beta}][\log \frac{\exp(l(a) + \beta\hat{\delta}(a))/Z'(\beta)}{\exp(l(a))/Z}] +$$

$$\sum_{a=1}^{N} \frac{\exp(l(a)+\beta\hat{\delta}(a))}{Z'(\beta)}[\hat{\delta}(a) - \frac{\frac{d}{d\beta}Z'(\beta)}{Z'(\beta)}] \tag{21}$$

$$=\sum_{a=1}^{N} \frac{\exp(l(a)+\beta\hat{\delta}(a))}{Z'(\beta)}[\hat{\delta}(a) - \frac{1}{Z'(\beta)}\frac{dZ'(\beta)}{d\beta}][\beta\hat{\delta}(a) + \log\frac{Z}{Z'(\beta)}]+$$

$$\sum_{a=1}^{N} \frac{\exp(l(a)+\beta\hat{\delta}(a))}{Z'(\beta)}[\hat{\delta}(a) - \frac{1}{Z'(\beta)}\frac{dZ'(\beta)}{d\beta}] \tag{22}$$

$$=\beta\sum_{a=1}^{N} \frac{\exp(l(a)+\beta\hat{\delta}(a))}{Z'(\beta)}[\hat{\delta}(a) - \frac{1}{Z'(\beta)}\frac{dZ'(\beta)}{d\beta}]\hat{\delta}(a)+$$

$$[\log\frac{Z}{Z'(\beta)}+1]\sum_{a=1}^{N} \frac{\exp(l(a)+\beta\hat{\delta}(a))}{Z'(\beta)}[\hat{\delta}(a) - \frac{1}{Z'(\beta)}\frac{dZ'(\beta)}{d\beta}]. \tag{23}$$

Let $\pi'(a;\beta) = \frac{\exp(l(a)+\beta\hat{\delta}(a))}{Z'(\beta)}$, then

$$\frac{1}{Z'(\beta)}\frac{dZ'(\beta)}{d\beta} = \sum_{a=1}^{N}\frac{\exp(l(a)+\beta\hat{\delta}(a))\hat{\delta}(a)}{Z'(\beta)} = \sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a). \tag{24}$$

Incorporating Eq. 24 to Eq. 23, we have

$$\frac{dD_{\mathrm{KL}}(\pi'(\beta)\|\pi)}{d\beta} =\beta\sum_{a=1}^{N}\pi'(a;\beta)[\hat{\delta}(a) - \sum_{a'=1}^{N}\pi'(a';\beta)\hat{\delta}(a')]\hat{\delta}(a)+$$

$$[\log\frac{Z}{Z'(\beta)}+1]\sum_{a=1}^{N}\pi'(a;\beta)[\hat{\delta}(a) - \sum_{a'=1}^{N}\pi'(a';\beta)\hat{\delta}(a')] \tag{25}$$

$$=\beta[(\sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a)^2) - (\sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a))^2] \tag{26}$$

$$=\beta[(\sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a)^2)(\sum_{a=1}^{N}\pi'(a;\beta)) - (\sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a))^2]. \tag{27}$$

By Cauchy–Schwarz inequality, $(\sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a)^2)(\sum_{a=1}^{N}\pi'(a;\beta)) - (\sum_{a=1}^{N}\pi'(a;\beta)\hat{\delta}(a))^2 \geq 0$. Because $\beta \geq 0$, $\frac{dD_{\mathrm{KL}}(\pi'(\beta)\|\pi)}{d\beta} \geq 0$. □

Based on this lemma, we can use a binary search algorithm to find the desired β with efficiency. The algorithm is shown in Alg. 1. In this algorithm, we use two hyperparameters, BSN_1 and BSN_2, to control the number of iterations of the binary search algorithm. In our experiments, we set $BSN_1 = 55$ and $BSN_2 = 15$.

C Experimental Settings

In this section, we describe the detailed settings of MPE tasks in our experiments.

Spread: There are 3 agents and 3 landmarks in a 5×5 grid. Agents need to occupy all 3 landmarks at the same time. Agents can observe both its location and the relative location of other agents and all landmarks. For each landmark i, let d_i be the distance from the nearest agent. The reward is $-\sum d_i$ subtracting the number of collisions.

Gather: There are 3 agents and 1 landmark in a 5×5 grid. Agents need to gather at the landmark simultaneously to get a reward. Agents can observe both its location and the relative location of other agents and the landmark. The reward is the sum of the negative distance between every agent and the landmark.

Formation: There are 4 agents and 1 landmark in a 5×5 grid. Agents need to form a square whose center is the landmark. Agents can observe both its location and the relative location of other

Algorithm 1 Find β by Binary Search

Input: $l_k, \hat{\delta}, KL^U, BSN_1, BSN_2$
Output: β

$\pi_k := \text{softmax}(l_k)$
$l := 0, r := 1, bsn := 0$
for $i = 1$ to BSN_1 **do**
  $\pi'_k := \text{softmax}(l_k + r\hat{\delta})$
  $bsn := i - 1$
  if $D_{\text{KL}}(\pi'_k \| \pi_k) > KL^U$ **then**
    Break
  end if
  $r := r * 2$
end for
for $i = 1$ to $BSN_2 + bsn$ **do**
  $m := \frac{l+r}{2}$
  $\pi'_k := \text{softmax}(l_k + m\hat{\delta})$
  if $D_{\text{KL}}(\pi'_k \| \pi_k) > KL^U$ **then**
    $r := m$
  else
    $l := m$
  end if
end for
return l

agents and the landmark. For each agent i, let d_i be the distance from the landmark, and α_i be the angle formed by the x-axis and the segment connecting it and the landmark. When calculating the reward, we first rearrange the order of agents so that $\alpha_i \leq \alpha_{i+1}$. Let $\beta_i = \alpha_i - i\frac{2\pi}{4}$. The reward is $-\sum_{i=1}^{4}(|\beta_i - \bar{\beta}| + |d_i - \bar{d}|)$, where $\bar{\beta} = \frac{1}{4}\sum_{i=1}^{4}\beta_i$ and $\bar{d} = \frac{1}{4}\sum_{i=1}^{4} d_i$.

D More Experimental Results

Influence of the attack frequency and the KL-divergence upper bound: We provide more experiments under different attack budgets in Fig. 7 and Fig. 8. From these results, we can see that better attack performance can be obtained with a higher attack frequency or a larger KL upper bound for our methods. By contrast, random attacks are less affected. These results are in line with our observation in the main text that our method find a better attack direction.

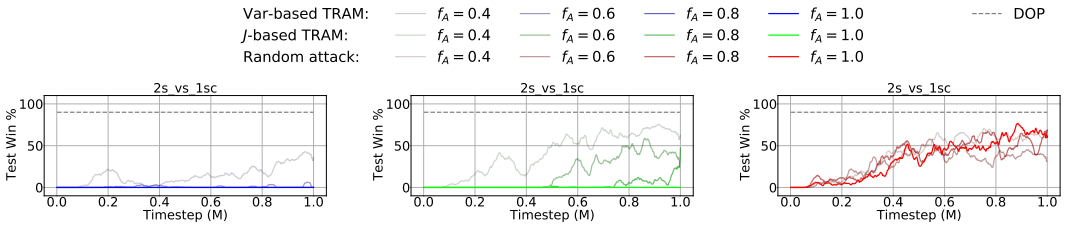


Figure 7: Performance of our methods and the random attack baseline under different attack frequencies on the map `2s_vs_1sc`. For these experiments, $KL^U = 6$.

We also show the log likelihood of being abnormal due to attacks for our methods and the random attack baseline under different attack frequencies and KL-divergence upper bounds in Table 2 and Table 3. A larger likelihood value in these tables indicates that the attacked policy is more like the intact policy. From these results, we can draw a similar conclusion as in the main text that our methods make a better use of the limited attack budget.

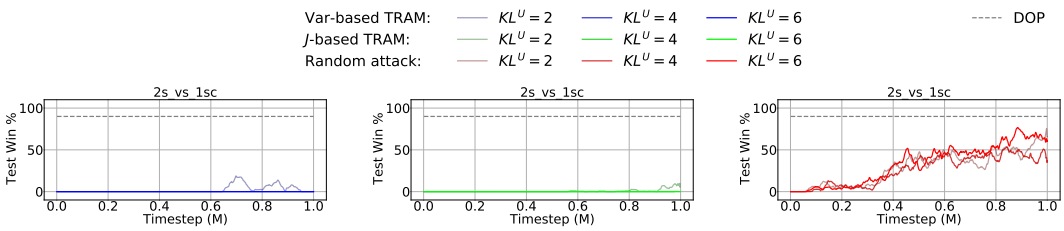


Figure 8: Performance of our methods and the random attack baseline under different KL-divergence upper bounds on the map `2s_vs_1sc`. For these experiments, $f_A = 1$.

Table 2: Log likelihood under different attack frequencies. For these experiments, $KL^U = 6$.

map	2s3z				3m				6m				2s_vs_1sc			
Frequency	0.4	0.6	0.8	1.0	0.4	0.6	0.8	1.0	0.4	0.6	0.8	1.0	0.4	0.6	0.8	1.0
Var-based TRAM	−1.24	−1.90	−2.31	−3.41	−1.23	−1.76	−2.13	−3.87	−1.12	−2.21	−3.01	−4.32	−1.89	−2.79	−3.27	−3.36
J-based TRAM	−0.72	−0.96	−1.27	−1.65	−0.80	−1.13	−1.67	−2.43	−1.03	−1.42	−1.81	−2.88	−0.98	−1.34	−1.82	−3.08
Random attack	−0.74	−1.03	−1.35	−1.76	−0.77	−1.05	−1.28	−1.53	−0.97	−1.35	−1.75	−2.28	−1.09	−1.53	−1.96	−2.45

Attacking other MARL algorithms: We change the attack target algorithm from DOP to another policy-based MARL algorithm, COMA [9] and show the result of both our methods and random attack on several SMAC maps in Fig. 9. From these results we can see that on COMA, Var-based TRAM is more effective than J-based TRAM, and random attack is the weakest attack.

TRAM on higher dimensional environments: We compare our methods with the random attack baseline (all attacking DOP) on two environments with higher dimension, 15m and 20m, to justify the scalability of our methods. The result is shown in Fig. 10. Var-based TRAM has the most significant influence, and J-based TRAM also undermines the performance a lot. The random attack baseline has little influence on DOP and even slightly improves the training performance on `20m`.

Results with confidence intervals of experiments under different attack budgets In Fig. 5 and Fig. 6, because curves are distinguished from each other by transparencies and showing confidence intervals may make some curves unclear, we hide the confidence intervals. We show these two figures with confidence intervals in Fig. 11 and Fig. 12.

E Limitations and Future Directions

In our work, we make an approximation for the first term of Eq. 8, $\nabla_{\tilde{\boldsymbol{\pi}}}\rho(s, \boldsymbol{a})$. This can lead to an inaccurate value of $\hat{\delta}$ computed by our methods. This might be exaggerated especially for long-horizon tasks, because a contaminated action may change the experience distribution of all following timesteps. One way to alleviate this issue can be training an FDM (forward dynamics model) for the environment to accurately model the change of state-action distribution caused by policy changes.

Another limitation of our method is that in both attack methods, we use the critic (or advantage function A) and the policy π to compute $\hat{\delta}$. This makes an additional assumption that the agents' critic and policy should be known. In some realistic scenarios where this information is not available, our attack methods will not be effective. One way to solve this issue can be using imitation learning to approximate the policy and the advantage function.

The two methods proposed in this work are actually maximizing one optimization goal:

$$E_{\rho(s,\boldsymbol{a}|\delta)}[|g(s,\boldsymbol{a}) - \nabla_{\theta^t} J(\theta_t)|^2] = E_{\rho(s,\boldsymbol{a}|\delta)}[|g(s,\boldsymbol{a})|^2 - 2g(s,\boldsymbol{a})^\top \nabla_{\theta^t} J(\theta_t) + |\nabla_{\theta^t} J(\theta_t)|^2]$$

Table 3: Log likelihood under different KL upper bounds. For these experiments, $f_A = 1$.

map	2s3z			3m			6m			2s_vs_1sc		
KL^U	2	4	6	2	4	6	2	4	6	2	4	6
Var-based TRAM	−2.09	−2.50	−3.41	−2.43	−3.74	−3.87	−2.82	−4.20	−4.32	−3.01	−3.23	−3.36
J-based TRAM	−1.13	−1.65	−1.65	−1.53	−2.48	−2.43	−1.68	−2.76	−2.88	−1.67	−3.06	−3.08
Random attack	−1.21	−1.54	−1.76	−1.26	−1.58	−1.53	−1.26	−2.16	−2.28	−1.74	−2.38	−2.45

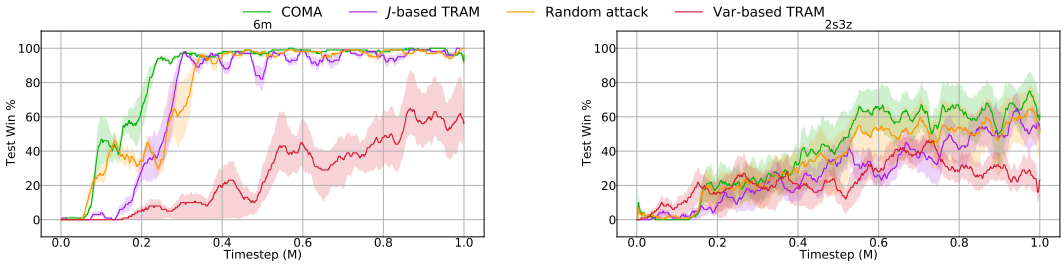


Figure 9: Performance of our methods and a random attack baseline attacking COMA on SMAC maps.

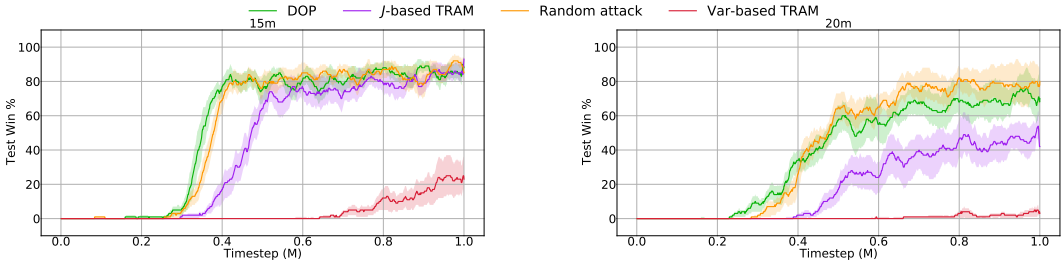


Figure 10: Performance of our methods and a random attack baseline (attacking DOP) on higher dimensional environments.

, where $g(s, \boldsymbol{a}) = A(s, \boldsymbol{a}) \nabla_{\theta^t} \log \boldsymbol{\pi}_{\theta^t}(\boldsymbol{a}|\boldsymbol{\tau})$ and $\rho(\cdot|\delta)$ follows the same definition as $\rho(\cdot)$ in the main text (just to emphasize the influence of δ to ρ). The variance-based method increases the first term ($E_{\rho(s,\boldsymbol{a}|\delta)}[|g(s, \boldsymbol{a})|^2]$) on the right hand side (RHS), the J-based method decreases the second term ($E_{\rho(s,\boldsymbol{a}|\delta)}[g(s, \boldsymbol{a})^\top \nabla_{\theta^t} J(\theta_t)]$), and the third term ($E_{\rho(s,\boldsymbol{a}|\delta)}[|\nabla_{\theta^t} J(\theta_t)|^2]$) is not affected by δ. Intuitively, maximizing this optimization goal increases the expected distance between the contaminated and the original policy gradients. Therefore, another future direction could be to investigate the attack when our methods are incorporated.

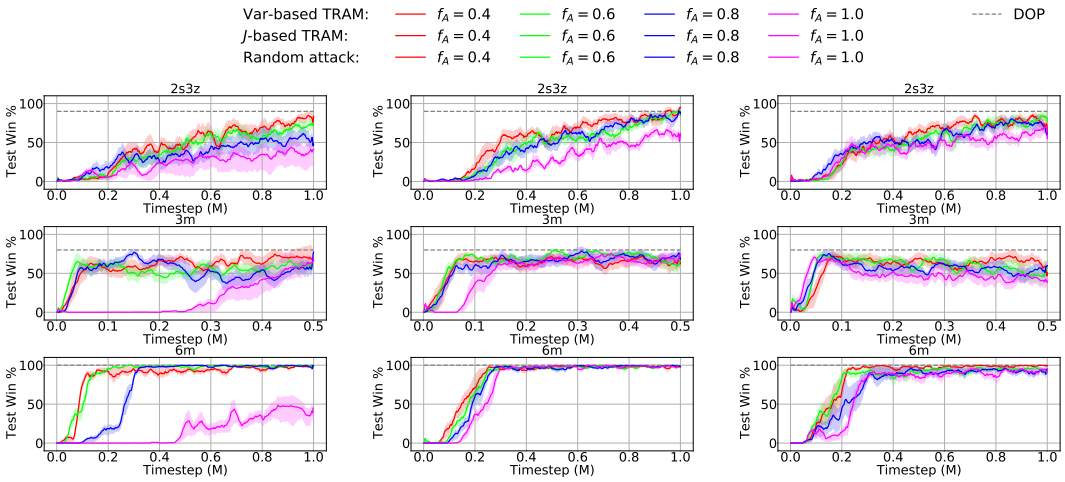


Figure 11: Include confidence intervals of Fig. 5.

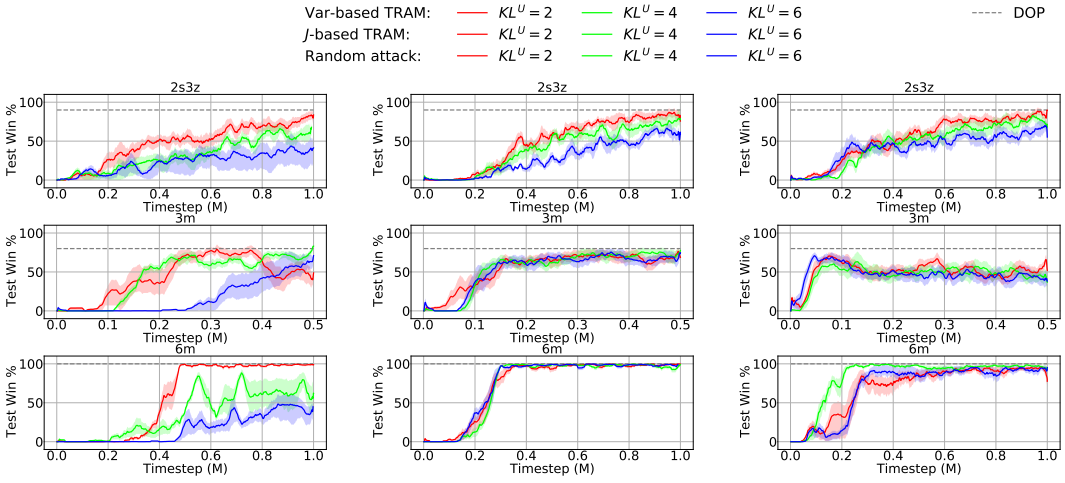


Figure 12: Include confidence intervals of Fig. 6..