Paper accepted for publication at ECAI 2025.

Distributed Detection of Adversarial Attacks in Multi-Agent Reinforcement Learning with Continuous Action Space

Kiarash Kazari[a,*], **Ezzeldin Shereen**[a] **and György Dán**[a]

[a]KTH Royal Institute of Technology, Stockholm, Sweden

Abstract. We address the problem of detecting adversarial attacks against cooperative multi-agent reinforcement learning with continuous action space. We propose a decentralized detector that relies solely on the local observations of the agents and makes use of a statistical characterization of the normal behavior of observable agents. The proposed detector utilizes deep neural networks to approximate the normal behavior of agents as parametric multivariate Gaussian distributions. Based on the predicted density functions, we define a normality score and provide a characterization of its mean and variance. This characterization allows us to employ a two-sided CUSUM procedure for detecting deviations of the normality score from its mean, serving as a detector of anomalous behavior in real-time. We evaluate our scheme on various multi-agent PettingZoo benchmarks against different state-of-the-art attack methods, and our results demonstrate the effectiveness of our method in detecting impactful adversarial attacks. Particularly, it outperforms the discrete counterpart by achieving AUC-ROC scores of over 0.95 against the most impactful attacks in all evaluated environments.

1 Introduction

In recent years we have witnessed enormous success for multi-agent reinforcement learning (MARL) algorithms, with potential applications in autonomous driving, 5G networks, robotics, and smart grid control [3]. Based on sequences of observations and rewards from the environment, cooperative MARL (c-MARL) enables a group of agents to perform complex sequential tasks by learning an optimal distributed policy. Besides enabling distributed execution of tasks, c-MARL has been shown to consistently outperform centralized and single-agent RL approaches, especially for complex tasks [3].

Despite the apparent success of c-MARL, its adoption in safety-critical applications will be dependent on its resilience to noise, faults, and adversarial manipulations. In particular, motivated by the rise of adversarial evasion attacks in machine learning and computer vision [9], the trustworthiness of seemingly high-performing data-driven algorithms is becoming increasingly important. In the context of c-MARL for example, an adversary that could compromise an agent could cause it to take sub-optimal actions (either through direct manipulation, or indirectly through manipulating its observations), so as to minimize the team reward [19]. One approach to address adversarial manipulations in MARL is to improve its resilience, e.g., using robust training through dataset augmentation or adversarial regularization [2]. Nonetheless, even though this can mitigate the effect of adversarial manipulations to some extent, it does not provide situational awareness.

Situational awareness requires the timely detection of adversarial activity, and allows to remove or replace misbehaving agents in real-time, or to adjust the policy of non-compromised agents. Existing anomaly detection schemes for single-agent RL [31, 45, 12, 38] are centralized and primarily focus on self-diagnosis by detecting manipulations in the sequence of observations or states. However, in many MARL applications, the agents' observations are private, hence these approaches are ineffective, especially against a powerful adversary capable of not only manipulating observations but also the victim's actions. Moreover, applying these approaches in a centralized manner to MARL would be limited to detecting only the presence of an attack, without being able to identify the specific victim.

Decentralized detection schemes for MARL, on the other hand, assume that the agents' action spaces are discrete and assign probabilities to all actions of the agents [21, 15]. However, many real-world applications of c-MARL involve agents operating in continuous action spaces. For instance, in robotics, agents often need to control motors or actuators with continuous control signals [3], or need to set voltage and frequency for the control of inverter-based resources in smart grids [41]. Adapting discrete-action schemes to continuous action spaces would require one to quantize the action space. Importantly, achieving acceptable detection accuracy would demand many quantization levels per action dimension; otherwise, critical information about the structure and different classes of actions in continuous space can be lost [46]. The alternative, assuming independence among different action dimensions is a strong assumption that often does not hold [5]. Thus, fine-grained quantization is necessary, but it leads to a complexity growing exponentially with the number of action dimensions. As a result, in high-dimensional action spaces, which are common in applications such as robotic control [29], existing detection methods become effectively impractical. Instead, a low-complexity method is needed specifically designed for MARL with continuous action spaces to detect adversarial attacks effectively.

In this paper, we propose a decentralized approach for detecting adversarial attacks in model-free c-MARL with continuous action space. Our contributions are as follows:

1. We utilize the observations of agents to train neural networks that predict the actions of other agents as parameterized multivariate Gaussian distributions.

* Corresponding Author. Email: kkazari@kth.se.

2. We use the predicted distribution for computing a normality score, which quantifies how well individual agents conform to their predicted behavior. We provide an analytical characterization of the mean normality score, which allows us to cast attack detection as a mean shift detection problem, and we propose to use the CUSUM method as a solution.
3. Through extensive simulations, we demonstrate the effectiveness of the proposed scheme in detecting state-of-the-art attacks in four benchmark MARL environments with continuous action space. Furthermore, we show that our proposed method outperforms the state-of-the-art discrete-action alternative in terms of performance and computational complexity.

The rest of this paper is organized as follows. Section 2 discusses previous work on adversarial attacks against MARL and countermeasures. Section 3 presents the system model and formulates the problem of decentralized attack detection. Section 4 presents the proposed detection approach. The performance of the proposed method is evaluated in Section 5. Section 6 concludes the paper.

2 Related Work

Many studies addressing adversarial attacks in single-agent reinforcement learning focus on perturbing states or observations. In these studies, the adversary manipulates observations so that the agent makes sub-optimal decisions with respect to the actual state of the environment [27],[1], [28], [24]. In the context of RL with continuous action spaces, [34] proposed real-time adversarial state manipulation attacks against Deep RL and showed the effectiveness of such attacks in MuJoCo [37] physics-based tasks like Walker2d.

In the domain of MARL, [19] considered perturbing the observations of a single agent in a c-MARL system. In [6], the authors proposed state manipulation adversarial attacks on cluster-based, heterogeneous MARL systems. More powerful attack models in the literature attack directly the actions of the agents. In the competitive MARL setting, [8] showed that an adversary could deceive an agent by manipulating the actions of the competing agent. [10] explored the robustness of state-of-the-art c-MARL algorithms against attacks on observations and actions. In a closely related work, [7] studied the effect of adversarial attacks on consensus-based networked MARL algorithms. In a different application domain, [33] investigated the vulnerability of MARL-based microgrid voltage control to adversarial attacks against sensor readings. Finally, [44] considered robustness against attacks to messages in communicative MARL.

Related to ours are previous works on anomaly detection for sequential data [25], [23],[39]. However, anomaly detection in the context of RL has received less attention compared to domains like video, audio, and text. Among the works addressing anomaly detection in RL, [45] introduced a framework for detecting anomalous state observations based on a Gaussian approximation of the state representation space. [31] proposed an entropy-based anomaly detector for identifying out-of-distribution observations, though not within an adversarial setting. Adversarial detection, i.e., identifying the adversarial samples generated in the state space of deep RL is the focus of [20]. Notably, none of these works addressed anomaly detection in a multi-agent setting. Applying these single-agent schemes to MARL would either necessitate centralized tracking of all agents or implementing anomaly detection locally for each agent. The former may not be feasible in decentralized multi-agent systems due to the resource-intensive nature of collecting data from all agents. The latter is sub-optimal as it does not account for adversaries with full control over the actions of the victim agent. In contrast, our approach involves other agents interacting with the environment for anomaly detection. Regardless of whether the misbehavior of the victim agent results from perturbations in its observations or actions, our method can detect anomalies.

Concerning multi-agent systems, existing works [32, 43] have proposed decentralized defenses against adversarial attacks in control systems. However, these approaches rely on a known model of the system, which is typically not available in MARL applications. Closest to our work are recent works addressing anomaly detection in c-MARL [15, 21]. These works rely on a categorical characterization of the actions taken by agents, making them applicable only to MARL problems with discrete action space. On the contrary, in this work we propose a detector that is applicable to MARL problems with continuous action space.

3 System Model and Problem Formulation

3.1 System Model

We consider a cooperative multi-agent environment modeled by a decentralized partially-observable Markov decision process (Dec-POMDP). The Dec-POMDP can be described by a tuple $M = (\mathcal{K}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{K}}, R, P, \{\mathcal{O}^i\}_{i\in\mathcal{K}}, \gamma)$, where $\mathcal{K} = \{1, 2, ..., K\}$ is the set of agents, $\mathcal{S}$ is the state space, and $\mathcal{O}^i$ is the set of observations of agent i. We let $\mathcal{A}^i \subseteq \mathbb{R}^{d_i}$ be the continuous action space of agent i. Moreover, R and P, denote the reward function, and the state transition probability, respectively, and $\gamma \in (0, 1)$ is a discount factor. At each time step $t \geq 0$, agent i observes $o_t^i \in \mathcal{O}^i$ from the environment and chooses an action $a_t^i \in \mathcal{A}^i$. The joint action profile $\mathbf{a}_t = \{a_t^i\}$ causes the state of the system to change from $s_t = s$ to $s_{t+1} = s'$ with probability $P(s'|s, \mathbf{a}_t)$, and a shared reward $R_t = R(s_t, \mathbf{a}_t)$ is received by all agents. In the reinforcement learning setting, which is the focus of this work, it is assumed that P and R are unknown functions. The objective of the agents is to maximize the total discounted average reward, i.e., $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t]$.

We say that agent j is an *observable neighbor* of agent i if agent i has access to the sequence of actions played by agent j. Note that this does not necessarily mean that those actions are in o_t^i. Instead, in many real-world applications, such as multi-UAV navigation or autonomous driving, where other agents' positions are part of an agent's observation, their past actions can be inferred (i.e., they are implicitly observable). We denote by $\mathcal{K}^i$ the set of all observable neighbors of agent i. We assume that $\mathcal{K}^i$ is constant over time, and that for each agent j there exists at least one agent i such that $j \in \mathcal{K}^i$, which is a prerequisite for distributed detection.

We consider that all agents have been trained using a MARL algorithm, and in the deployment time, agent i acts based on a local policy $\pi^i(\tau_t^i)$, where $\tau_t^i \in \Gamma^i \triangleq (\mathcal{O}^i \times \mathcal{A}^i)^*$ is the history of action-observations of agent i up to time t, i.e., $\tau_t^i = (o_0^i, a_0^i, ..., o_t^i)$. $\pi^i(\tau_t^i)$ can be a stochastic or a deterministic policy.

3.2 Attack Model

We consider an adversary that can eavesdrop on the observations and can manipulate the actions of one agent $v \in \mathcal{K}$ in the deployment time of c-MARL. That is, at each time step t, the adversary can observe $o_t^{adv} = o_t^v$ from the environment, and can cause the victim agent to take an action $a_t^{adv} \in \mathcal{A}^v$ instead of a_t^v, where a_t^{adv} follows the adversary's policy $\pi^{adv} \neq \pi^v(\tau_t^v)$. We refer to agent v as the victim agent. This attack model is in line with the models considered in

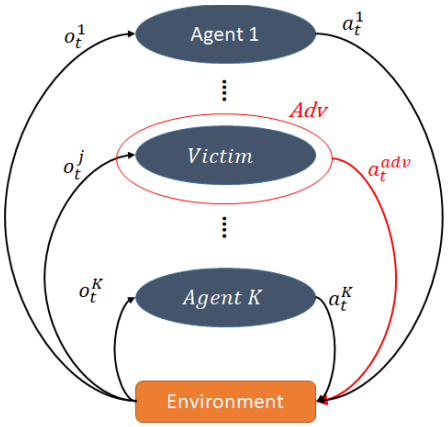


Figure 1. System model with agent j as victim.

[10] and [15], and is illustrated in Figure 1. Note that although we assume that only one agent is compromised, our proposed approach is applicable in scenarios involving multiple victims (c.f. Section 5.8).

3.3 Problem Formulation

Consider that the adversary starts to attack some agent v at time step t_0 (t_0 might be considered ∞ in case that no attack happens). Given the sequence of actions of all agents, our objective is to identify the victim agent as soon as possible after the attack starts in a distributed manner, i.e., detection should be done by the agents based on locally available information. Consequently, treating agent i as an observer and agent $j \in \mathcal{K}^i$ as a prospective victim, the considered anomaly detection problem can be formulated as follows.

Decentralized c-MARL Attack Detection: Given a history $\tau_t^{ij} = o_0^i, a_0^j, o_1^i, a_1^j, ..., o_t^i, a_t^j$ at time t, find a function $M^{ij} : (\mathcal{O}^i \times \mathcal{A}^j)^* \rightarrow \{0, 1\}$, s.t. $M^{ij}(\tau_t^{ij}) = 0$ if the sequence τ_t^{ij} is normal (i.e., it is the consequence of following π^j by agent j) and $M^{ij}(\tau_t^{ij}) = 1$ if τ_t^{ij} is abnormal (agent j is a victim).

4 Decentralized Attack Detection

In this section, we describe our proposed detector, i.e., the algorithm for implementing function M^{ij} for agent $i \in \mathcal{K}$ and agent $j \in \mathcal{K}^i$. The proposed algorithm is composed of two steps. First, it estimates the distribution of agent j's action (for the upcoming time-step) based on the observations of agent i. In the second step, a normality score is computed based on the estimated distribution and the action actually taken by agent j; the normality score is then used for detection, relying on a characterization of its mean and standard deviation. Next, we present a detailed description of these two steps.

4.1 Learning Action Distributions

We propose to approximate the action distribution $f^{ij}(.|\tau_t^i)$ of agent j from agent i's point of view by a d_j-dimensional Gaussian distribution (recall that d_j is the dimension of the action space) parameterized by its mean, $\mu_t^{ij}(\tau_t^i) \in \mathbb{R}^{d_j}$, and its covariance matrix, $\mathbf{\Sigma}_t^{ij}(\tau_t^i) \in \mathbb{R}^{d_j \times d_j}$. Observe that the parameters of the distribution are a function of the history τ_t^i. The proposed approximation is motivated by that many deep RL algorithms for continuous action spaces represent a policy as a multivariate Gaussian-distribution [42]. While this does not imply that the actions of other agents based on the observation of a single agent would be Gaussian, our results in Section 5 show that the Gaussian distribution can effectively approximate the other agents' behaviour based on local observations. The approximation can thus be formulated as

$$f^{ij}(\mathbf{x}|\tau_t^i) \approx (2\pi)^{-\frac{d_j}{2}} \det(\mathbf{\Sigma}_t^{ij})^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu_t^{ij})^\top (\mathbf{\Sigma}_t^{ij})^{-1}(\mathbf{x}-\mu_t^{ij})\right] = g(\mathbf{x}; \mu_t^{ij}, \mathbf{\Sigma}_t^{ij}). \quad (1)$$

The objective of agent i is to learn a function that takes τ_t^i as the input, and outputs μ_t^{ij} and $\mathbf{\Sigma}_t^{ij}$. We propose to approximate this function using a recurrent neural network (RNN), represented as NET^{ij}. Our approach is based on the intuition that the task of predicting the parameters of the Gaussian distribution can be regarded as a regression problem where (the sequence of) agent i's observations are the inputs. In the training phase of NET^{ij}, the objective is to find weights $\boldsymbol{\theta}^{ij}$ such that the outputs, i.e., $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ would maximize the log-likelihood of observing a_t^j given τ_t^i. Accordingly, NET^{ij} is trained to maximize the expected log-likelihood $\mathbb{E}[\log f^{ij}(a_t^j|\tau_t^i)]$. Using (1), the objective function can be expanded as

$$\max_{\boldsymbol{\theta}^{ij}} \mathbb{E}[\log f^{ij}(a_t^j|\tau_t^i)] = \max_{\boldsymbol{\theta}^{ij}} \mathbb{E}[\log g(a_t^j; \mu_t^{ij}, \mathbf{\Sigma}_t^{ij})] = \max_{\boldsymbol{\theta}^{ij}} \mathbb{E}\Big[-\frac{1}{2}\Big(\log\det(\mathbf{\Sigma}_t^{ij}) + (a_t^j-\mu_t^{ij})^\top(\mathbf{\Sigma}_t^{ij})^{-1}(a_t^j-\mu_t^{ij})\Big)\Big] \quad (2)$$

Note that $\mathbf{\Sigma}_t^{ij}$ is a covariance matrix, hence it should be symmetric and positive definite. In order to ensure this condition is met, we consider the Cholesky factorization [14] of the covariance matrix as $\mathbf{\Sigma}_t^{ij} = \mathbf{L}_t^{ij}\mathbf{L}_t^{ij\top}$ where $\mathbf{L}_t^{ij}$ is a lower triangular matrix. Consequently, we treat the non-zero elements of $\mathbf{L}_t^{ij}$ as the outputs of NET^{ij}, which are unconstrained. Now, the objective function in (2) is equivalent to

$$\min_{\boldsymbol{\theta}^{ij}} \mathbb{E}\Big[2\log\det(\mathbf{L}_t^{ij}) + (a_t^j-\mu_t^{ij})^\top(\mathbf{L}_t^{ij}\mathbf{L}_t^{ij\top})^{-1}(a_t^j-\mu_t^{ij})\Big] \quad (3)$$

$$= \min_{\boldsymbol{\theta}^{ij}} \mathbb{E}\Big[2\sum_{k=1}^{d_j}\log l_t^{ij}[k] + y_t^{ij\top}y_t^{ij}\Big] \quad (4)$$

where $l_t^{ij}[k]$ denotes the k-th diagonal element of $\mathbf{L}_t^{ij}$ and $y_t^{ij} \triangleq \mathbf{L}_t^{ij^{-1}}(a_t^j - \mu_t^{ij})$. Note that since $\mathbf{L}_t^{ij}$ is a lower triangular matrix, there is no need for inverting it and y_t^{ij} can be computed directly using low complexity algorithms like forward substitution [14]. Training NET^{ij} is done by collecting samples of the form (τ_t^i, a_t^j) during the normal execution of the policies $\{\pi^j\}_{j\in\mathcal{K}}$ of the agents, and then optimizing the network's parameters according to the objective (4).

4.2 Normality Score and Detection Procedure

We use the action density functions learned in the previous step for quantifying the normality of the sequence of agent j's actions up to time step t. While the problem seemingly resembles that of sequential hypothesis testing [17], where the objective is to determine whether a sequence of samples are generated from a given distribution, in our case the distribution is state-dependent and changes

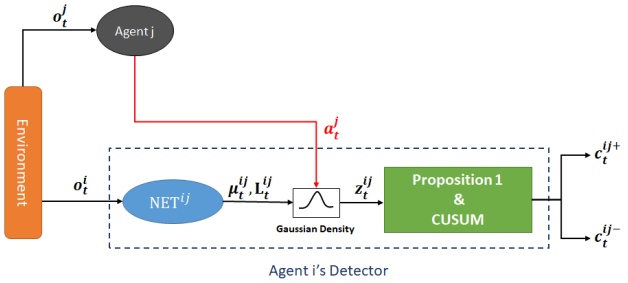


Figure 2. Proposed detection scheme for agent i as the observer and agent j as the potential victim.

over time. Hence, existing approaches to sequential hypothesis testing cannot be applied directly.

Instead, motivated by the log-likelihood ratio computed in sequential hypothesis testing approaches, we propose to use the normalized log-likelihood based on the approximated action distributions to compute a normality score defined as

$$z_t^{ij} = \log\left(\frac{f^{ij}(a_t^j|\tau_t^i)}{\max_{\mathbf{x}} f^{ij}(\mathbf{x}|\tau_t^i)}\right), \tag{5}$$

which represents the log-likelihood of the predicted density at a_t^j normalized by the maximum value of the density function. Intuitively, the reason for the normalization is to take the confidence of the predictions into account.

Importantly, if f^{ij} is Gaussian then z_t^{ij} is closely related to the Mahalanobis distance of a_t^j and $f^{ij}(.|\tau_t^i)$. To see why, observe that f^{ij} attains its maximum at $\mathbf{x} = \mu_t^{ij}$. Thus,

$$z_t^{ij} = \log\left[\frac{g(a_t^j; \mu_t^{ij}, \mathbf{\Sigma}_t^{ij})}{g(\mu_t^{ij}; \mu_t^{ij}, \mathbf{\Sigma}_t^{ij})}\right] \tag{6}$$
$$= -\frac{1}{2}(a_t^j - \mu_t^{ij})^\top (\mathbf{\Sigma}_t^{ij})^{-1}(a_t^j - \mu_t^{ij}) = -\frac{D(a_t^j, f_t^{ij})}{2},$$

where $D(.,.)$ denotes the Mahalanobis distance. An essential property of the score we propose for detection is the invariance of its expected value over time.

Proposition 1. *Consider agents $i, j \in \mathcal{K}$ at time t, where $j \in \mathcal{K}^i$, and assume $a_t^j|\tau_t^i \sim \mathcal{N}(\mu_t^{ij}, \mathbf{\Sigma}_t^{ij})$. Then the first and second moments of z_t^{ij} are*

$$m_z^j \triangleq \mathbb{E}[z_t^{ij}] = -d^j/2, \quad {\sigma_z^j}^2 \triangleq \mathbb{E}[(z_t^{ij} - m_z^j)^2] = d^j/2. \tag{7}$$

Proof. For ease of notation, we drop the subscript t and superscripts i, j. We can write

$$\begin{aligned}
\mathbb{E}[z] &= -\frac{1}{2}\mathbb{E}[(a-\mu)^\top \Sigma^{-1}(a-\mu)] \\
&= -\frac{1}{2}\mathbb{E}[\text{tr}((a-\mu)^\top \Sigma^{-1}(a-\mu))] \\
&= -\frac{1}{2}\mathbb{E}[\text{tr}(\Sigma^{-1}(a-\mu)^\top (a-\mu))] \\
&= -\frac{1}{2}\text{tr}(\Sigma^{-1}\mathbb{E}[(a-\mu)^\top (a-\mu)]) \\
&= -\frac{1}{2}\text{tr}(\Sigma^{-1}\Sigma) = -d/2.
\end{aligned} \tag{8}$$

For the variance, we can write

$$\mathbb{E}[z^2] = \frac{1}{4}\mathbb{E}[((a-\mu)^\top \Sigma^{-1}(a-\mu))^2]. \tag{9}$$

Since Σ is a positive definite matrix, it has a unique symmetric positive semi definite square root R, such that $RR = \Sigma$. Then for $x = R^{-1}(a-\mu)$ it holds that $x \sim \mathcal{N}(0, I)$. Thus, we have

$$\begin{aligned}
\mathbb{E}[z^2] &= \frac{1}{4}\mathbb{E}[(x^\top R R^{-1} R^{-1} R x)^2] = \frac{1}{4}\mathbb{E}[(x^\top x)^2] \\
&= \frac{1}{4}(\text{tr}(I)^2 + 2\text{tr}(I)) = \frac{1}{4}(d^2 + 2d).
\end{aligned} \tag{10}$$

Accordingly, $\sigma_z^2 = \mathbb{E}[z^2] - \mathbb{E}[z]^2 = d/2$. □

Proposition 1 shows that, although the distribution of predicted actions varies over time, as long as the actions of agent j follow the conditional distribution expected by observer agent i, the expected value of the normality score remains constant.

Importantly, Proposition 1 allows us to cast attack detection as the problem of detecting a shift in the mean of a sequence of random variables, for which CUSUM is a well-known stopping rule [26], which is optimal under the assumption that the samples are i.i.d [16, 40]. In our setting, the samples z_t^{ij} are not independent; however, the complexity of their dependencies makes it infeasible to derive an optimal stopping rule. We thus propose to apply CUSUM on the normality score z_t^{ij} combined with Proposition 1 for detection. As shown in Section 5, it performs well despite samples not being independent. Accordingly, our proposed detection method is based on computing

$$c_t^{ij^+} = \max\{0, c_{t-1}^{ij^+} + \frac{z_t^{ij} - m_z^j}{\sigma_z^j} - w\}, \tag{11}$$

$$c_t^{ij^-} = \max\{0, c_{t-1}^{ij^-} + \frac{m_z^j - z_t^{ij}}{\sigma_z^j} - w\}, \tag{12}$$

which are used to determine a deviation from the mean in the positive and the negative directions, receptively, w is a tunable hyperparameter that determines CUSUM's sensitivity, and $c_{-1}^{ij^+} = c_{-1}^{ij^-} = 0$. For detection, each agent i continuously updates $c_t^{ij^+}$ and $c_t^{ij^-}$ corresponding to every agent j in $\mathcal{K}^i$, and considers agent j as compromised if either $c_t^{ij^+} > \beta^{ij^+}$ or $c_t^{ij^-} > \beta^{ij^-}$, where β^{ij^+} and β^{ij^-} are predefined thresholds. We refer to the proposed detection scheme as the Parameterized Gaussian CUSUM (PGC) detector. Figure 2 shows the overall structure of the proposed detector.

The proposed detector can be combined with different decision rules for distributed detection of an attack. A straightforward rule would be to consider an agent as compromised once any of the other agents detects it as attacked. Alternatively, one could consider an agent compromised when at least $u > 1$ agents detect that it is compromised.

5 Evaluation

We present results obtained using the proposed detector against four types of attacks in four MARL environments.

5.1 MARL Environments

We used four PettingZoo c-MARL continuous action environments [35] for the evaluation, namely, *Multiwalker* [11], *Tag* (Simple Tag) [22], *World Comm* [22], and *Pistonball*. The main properties of the considered scenarios are summarized in Table 1. Based on the observations provided by the environments, in Pistonball and Multiwalker, $\mathcal{K}^i$ is the set of the agents adjacent to agent i, while in the other two environments, $\mathcal{K}^i$ includes all agents other than i.

Table 1. Number of agents, observation size, action space dimension, and time limit in the considered scenarios. In Tag and World Comm one of the agents does not belong to the c-MARL team.

Environment	Agents	Obs. Size	Action Dim.	Time Limit
Multiwalker	5	31	4	200
Tag	4	16	5	25
WorldComm	5	25	9	25
Pistonball	10	RGB (84,84,3)	1	125

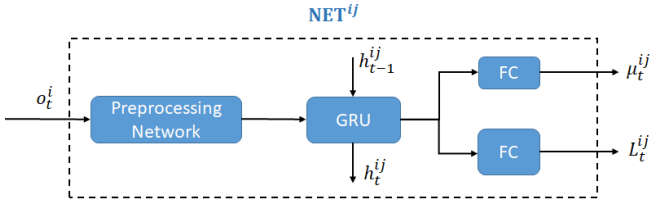


Figure 3. Structure of NET^{ij} used for prediction.

5.2 *Attack Strategies*

We use four attack strategies for the evaluation.

Random Attack (RAND): A naive adversary that at each step chooses an action uniformly at random in $\mathcal{A}^v$.

Action Attack (ACT): The adversary's objective is to minimize the team reward. The policy of the adversary is obtained by solving a single-agent reinforcement learning problem with the action space defined as $\mathcal{A}^v$ and the objective of minimizing $\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t R_t]$ [10]. Since the policies of the non-victim agents are fixed, they can be considered part of the environment.

Projected Gradient-Based Attack (GRAD): This approach is commonly used in single agent RL to manipulate agent's observations [27]. It is based on finding a suitable l_∞-bounded manipulation through a perturbation in the gradient-ascent direction of a loss function, e.g., as in FGSM [9]. Due to continuity of the action space, we apply this method directly to the actions. Accordingly, we considered the following manipulation strategy based on the Q function:

$$a_t^{adv} = Proj\{a_t^v - \epsilon sign(\nabla_a Q^v(\tau_t^v, a_t^v))\}, \qquad (13)$$

where $Q^v(\tau, a)$ is the victim's Q function, and $Proj$ denotes the projection onto $\mathcal{A}^v$.

Dynamic Adversary (DYN): The dynamic adversary is a powerful adversary that is aware of the detection scheme and can observe the normality scores that the agents compute. It uses this information for learning attacks that are hard to detect using the proposed detector, as done in [15]. Following the notion of an expectedly undetectable attack introduced in [15], the adversary's problem can be formulated as a constrained RL problem whose solution would be a Markovian policy. To tailor this attack against our detection procedure, we consider the reward function for this single agent RL as $R_t^{adv} = -(R_t + \sum_{i:v\in\mathcal{K}^i}\lambda_i|z_t^{iv} - m_z^v|)$, where λ_i is a weight parameter that captures the importance of remaining undetected by agent i, and effectively controls the trade-off between detectability and the impact of the attack on the team reward.

For instance, for $\lambda_i = 0$ the attack is equivalent to ACT, maximizing attack impact. Conversely, for high values of λ_i, the objective is dominated by trying to remain undetected by keeping the normality score close to its expectation. For simplicity we consider uniform weights, i.e., $\lambda_i = \lambda$.

5.3 *Baselines*

Existing single agent RL anomaly detection methods typically detect anomalies in the observations or states of an agent [31, 45]. Such detection schemes could be used by the victim, but if used by non-victim agents, they would not be able to identify the victim agent, only the fact that there may be an attack. To provide a fair comparison, we thus focus on baselines that allow to identify victim agents. We provide a comparison to [45] in Appendix C.4.

As baselines we thus use the detection method proposed in [15] for MARL with discrete action spaces, which we refer to as *Discrete*, and a number of variants of our detection method. To adapt *Discrete* to a continuous action space, we uniformly quantize each dimension of the action space into Q intervals, resulting in an action set of size Q^d if the continuous action set has dimension d.

We consider three variants of our detection method: using the Dirichlet distribution, the Beta distribution, and Gaussian with independent components (i.e., a diagonal covariance) as the distribution approximating f^{ij}. These variants are referred to as *Dirichlet*, *Beta*, and *Independent PGC (I-PGC)* in this section. For the *Dirichlet* and *Beta* baselines, since the normality score is not constant (c.f. Proposition 1), CUSUM is not applicable. Instead, we used the window-average-based method proposed by [15] to compute the decision metrics. For comparison, we also include results for replacing CUSUM with this metric in PGC in Appendix C.3.

5.4 *Evaluation Methodology*

Given the four environments and four attack strategies, we followed four steps for the evaluation.[1]

5.4.1 *Step1: Training the team agents*

We used RLlib Python framework [18] to train the c-MARL agents. For Multiwalker and Pistonball, the multi-agent implementation of the PPO algorithm [30], and for Tag and World Comm, the multi-agent APE-X DDPG algorithm [13] were used. The selection of the algorithms was based on the results reported in [36].

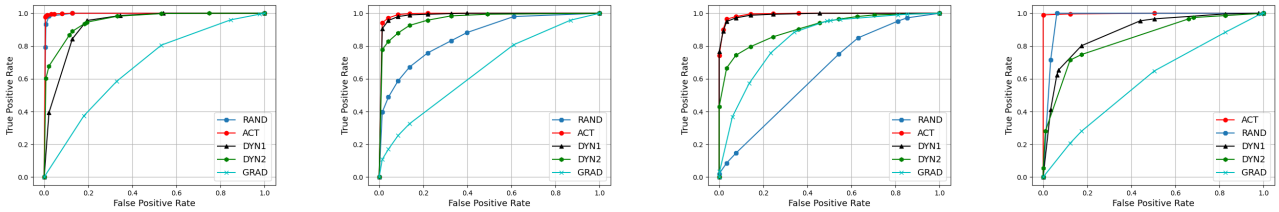
5.4.2 *Step2: Training NET^{ij}*

The considered structure for each NET^{ij} is shown in Figure 3. In Pistonball, we used the initial convolutional layers of the agents' policy networks, obtained during the c-MARL training stage, as the preprocessing network. In other environments, the preprocessing network consists of a fully connected layer followed by a ReLU activation, trained from scratch along with the other layers. We provide further details in Appendix D. We trained NET^{ij} over 5000 episodes for Multiwalker and Pistonball, and 20,000 episodes for Tag and World Comm. During the mentioned training phase the team agents used the policies learned in Step 1.

5.4.3 *Step3: Training the Attacks*

We trained the ACT and the DYN attacks using a single-agent RL algorithm as explained in Section 5.2. We trained multiple DYN attacks using different λ values. For training each attack, we employed the single-agent PPO algorithm in RLlib over 20,000 episodes and selected the policy with the best adversarial reward.

[1] Code available at https://github.com/kiarashkaz/PGC-Adversarial-Detection-Continuous



(a) Multiwalker (b) Tag (c) World Comm (d) Pistonball

Figure 4. ROC of the proposed method against different attacks in all environments.

Table 2. AUC score of PGC and Discrete against RAND, ACT, GRAD and DYN attacks in all environments.

	Multiwalker		**Tag**		**World Comm**		**Pistonball**	
Attack Types	*PGC*	*Discrete*	*PGC*	*Discrete*	*PGC*	*Discrete*	*PGC*	*Discrete*
ACT	**0.996**	0.972	**0.993**	0.948	**0.995**	0.821	**0.999**	0.758
RAND	**0.995**	0.855	0.843	**0.893**	0.677	**0.713**	**0.997**	0.970
GRAD	**0.674**	0.566	0.653	**0.858**	0.884	**0.913**	**0.581**	0.554
DYN1	**0.929**	0.818	**0.988**	0.964	**0.992**	0.754	**0.907**	0.711
DYN2	**0.954**	0.788	**0.968**	0.944	**0.912**	0.707	**0.876**	0.658

Table 3. Average total episodic reward under different attacks.

	Multiwalker	Tag	World Comm	Pistonball
No Attack	-12.7	101.8	37.6	228.6
ACT	-107.6	64.9	26.7	83.1
RAND	-75.6	68.1	30.4	202.1
Grad	-42.7	90.4	34.7	215.1
DYN1	-96.9	65.1	27.8	95.5
DYN2	-89.6	69.2	30.1	139.3

5.4.4 Step4: Evaluation of the Detector

For each attack (and also a scenario without any attack), we used the c-MARL polices, an attack policy, and a detector for 500 episodes. **The detection rule** applied in our evaluations was as follows: if any of the non-victim agents triggered a detection, we considered the attack as detected at that particular time step (i.e., $u = 1$). For the evaluation we considered the same set of detection thresholds for all agents. The value of hyperparameter w is 0.5 in Multiwalker and World Comm, 0.75 in Tag and 0.05 in Pistonball. We provide results for other values of w in Appendix C.1.

5.5 Detection Performance

We first use the Receiver Operating Characteristic (ROC) curve to evaluate the performance of the proposed detector. The ROC curve shows the true positive rate against the false positive rate, generated by varying detection thresholds β^{v^+} and β^{v^-}. The true positive rate is defined as the fraction of attacked episodes that are successfully detected, while the false positive rate is the fraction of unattacked episodes incorrectly classified as attacked. The detection performance can characterized using the Area Under the ROC Curve (AUC), where AUC=1 corresponds to a perfect detector. The ROC curves of all environments is shown in Figure 4.

Figure 4 and Table 2 show the ROC curves and the AUC scores of PGC against different attacks in all environments, respectively. DYN1 and DYN2 correspond to two DYN attacks trained with different values of λ, illustrating varying levels of detectability. To illustrate the corresponding attack impact, Table 3 shows the average total reward collected by the team agents under the different attacks. These results show that PGC can successfully detect the impactful attacks in all environments. The ACT attack, which causes the most significant reduction in team reward (e.g., from -12.7 to -107.6 in Multiwalker), can be detected almost perfectly, with an AUC score close to 1. Detecting attacks with lower impact is more challenging. For example, GRAD in Pistonball or RAND in World Comm can partially bypass the detector (resulting in low AUC scores), but at the same time, the reduction in the team reward caused by these attacks is negligible. One explanation for this observation is that the victim agent's original policy closely resembles these adversarial policies, making it unsurprising that distinguishing between the agent's normal behavior and these attacks is challenging. The trade-off between the attack impact and the detectability of the attack can be observed in other scenarios as well, which confirms that *only those attacks can (partially) remain undetected that have very limited impact on the performance of the system.*

Table 2 also shows that PGC outperforms *Discrete* in most cases, especially against the more impactful attacks. Discrete performs better than PGC against GRAD in some scenarios. This can be explained by the fact that manipulated actions resulting from Grad are actually slight variations of the normal actions at each step. Thus they might not trigger a high anomaly score in PGC, but they might correspond to a different beam when distributions are modeled over a discrete set, resulting in a relatively higher anomaly score.

Another key aspect of comparison is complexity. *PGC* has significantly lower computational cost than *Discrete*. For example, in Tag, the detector networks for *Discrete* require 243 outputs (one per action), while PGC requires only 20 outputs (one per action space dimension and one per covariance). This difference becomes even more pronounced in high-dimensional environments. For example, *Discrete* requires neural networks with $3^9 \approx 2 \times 10^4$ outputs, while this number is only 54 for PGC. The high output dimensionality of *Discrete* necessitates significantly larger hidden layers for effective training, making it extremely computationally demanding to achieve performance comparable to that of PGC.

Finally, an important metric for evaluating a detector's performance is its time to detection. We provide the evaluation results for PGC in Appendix C.2.

Table 4. AUC score of different detector baselines against RAND.

Variation	Multiwalker	Tag	World Comm
PGC	**0.99**	**0.84**	**0.67**
IPGC	0.73	0.53	0.55
Beta	0.51	0.50	0.50
Dirichlet	0.54	0.50	0.50

5.6 Why Multivariate Gaussian?

Recall that PGC uses a multivariate Gaussian distribution with a non-diagonal covariance matrix for the approximation of the action distribution. IPGC is a simpler alternative that assumes different dimensions of a_t^j are independent, and thus, $\mathbf{\Sigma}_t^{ij}$ is diagonal. Hence, NET^{ij} would have $2d$ outputs. While this simplification may not impact performance in certain scenarios, there are environments where the assumption of independence of action dimensions does not hold.

Table 4 shows the AUC scores of PGC and IPGC against RAND in environments with an action dimension greater than 1. The results indicate significantly lower scores for IPGC, particularly in Tag and World Comm, where RAND is almost undetectable by IPGC, despite its impact in Tag. The poor performance of IPGC is due to the high correlation between elements of the action vector in non-compromised agents. Training independent predictor networks in such cases results in a predicted distribution with high variance, making the RAND attack undetectable. We note that, however, there might be environments where the elements of the action vector are close to independent, and in such environments IPGC can be used to reduce the complexity of detection.

Table 4 shows the AUC scores obtained using Beta and Dirichlet distributions (which partially account for correlations between dimensions) against RAND. The results shows that neither distribution produces an effective detector, as the RAND attack is nearly undetected in all scenarios.

To further understand how accurate is the Gaussian approximation of the distributions f^{ij}, we compared the episodic average of the normality score under normal agent behavior with its expected value under the Gaussian assumption (Appendix A). We found that these quantities are very close in all environments, supporting that the Gaussian approximation may be accurate. Clearly, there may be MARL scenarios where f^{ij} are multi-modal, in which case a centralized approach could be a more effective choice for detection. We provide a more detailed discussion in Appendix A.

5.7 Parameter Sharing for Reduced Complexity

So far, we showed results when each agent has a separate detector for keeping track of every observable neighbor's action. This requires a total of $\sum_i |\mathcal{K}^i|$ predictor networks to be trained for detection,which can amount to as many as $K(K-1)$ networks in the worst case. In the following, we explore whether parameter sharing, a common approach in training MARL algorithms [4], [36], can be effectively applied in attack detection. With parameter sharing, a single predictor network is trained per potential victim agent, and the trained model is shared among all agents that observe that neighbor, reducing the total number of models to K. It is important to note that parameter sharing is applicable only to environments where the dimension of all agents' observations is the same, and there exists some form of symmetry among the observation spaces of the agents.

Table 5 shows the AUC score of PGC trained using parameter sharing. Comparing Table 2 and Table 5 shows that parameter sharing does not affect performance. Interestingly, in some cases, parameter sharing has even enhanced performance. This observation aligns with the findings presented by [36], which demonstrated that parameter sharing can enhance the training of MARL algorithms with continuous action spaces. The improvement is attributed to parameter sharing utilizing more trajectories to train a shared architecture, leading to potential improvements in sample efficiency.

Table 5. AUC score of PGC trained with parameter sharing against RAND and ACT.

	Multiwalker	Tag	World Comm	Pistonball
ACT	0.995	0.994	0.997	0.999
RAND	0.995	0.817	0.713	0.997

Table 6. Average total team reward and AUC scores of PGC when detecting ACT and RAND attacks against two victim agents.

	Multiwalker		Pistonball	
Attack Type	Reward	AUC	Reward	AUC
ACT	-109.6	0.998	75.8	0.997
RAND	-96.0	0.997	180.6	0.983

5.8 Multiple Victims

Finally, we consider the case when there are more than one victim agents in the environment. Recall that each predictor is trained using the non-attacked behavior of individual agents. Consequently, the number of victim agents should not influence the training of the predictors. Furthermore, the computation of the normality score for an agent's actions is independent of the actions of other agents. This independence in normality scores for different agents should enable PGC to perform well in the presence of multiple victim agents.

To illustrate this, we evaluated PGC against two victim agents in Multiwalker and Pistonball. We considered that detection occurs when both victims are detected. Table 6 shows the team reward and the AUC scores obtained. Overall, the results confirm that PGC can be utilized even when there is more than one victim.

6 Conclusion

In this paper, we introduced a decentralized approach for detecting adversarial attacks on multi-agent RL systems with continuous action space. Our proposed scheme leverages the observations of the agents to predict the action distributions of other agents. We used RNNs to approximate these distributions as parameterized Gaussian densities. We put forward a low-complexity anomaly score, computed by comparing the predictions with the actual actions taken by the agents. We analytically showed that the proposed score has a constant mean and employed the CUSUM method to detect deviations from this value. The proposed method was evaluated against various attack strategies with different levels of impact and detectability. Our results show that the proposed method outperformed its discrete counterpart, with only low-impact attacks having the potential to evade detection. Moreover, we empirically showed that the non-diagonal elements of the covariance matrix of the parameterized distributions used for detection are important for high detection performance. We also found that the computational complexity of training the detector could be reduced through parameter sharing without compromising the detection rate, and that the proposed detector is effective in scenarios with multiple victims.

Acknowledgements

This work was partly funded by the Swedish Research Council through projects 2020-03860 and 2024-05269, and by Digital Futures through the CLAIRE project. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at Linköping University partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

[1] V. Behzadan and A. Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *Proc. of International Conference on Machine Learning and Data Mining (MLDM)*, pages 262–275. Springer, 2017.

[2] A. Bukharin, Y. Li, Y. Yu, Q. Zhang, Z. Chen, S. Zuo, C. Zhang, S. Zhang, and T. Zhao. Robust multi-agent reinforcement learning via adversarial regularization: Theoretical foundation and stable algorithms. *Proc. of NeurIPS*, 2023.

[3] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11), 2021. ISSN 2076-3417. URL https://www.mdpi.com/2076-3417/11/11/4948.

[4] F. Christianos, G. Papoudakis, M. A. Rahman, and S. V. Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *Proc. of ICML*, pages 1989–1998, 2021.

[5] R. Dadashi, L. Hussenot, D. Vincent, S. Girgin, A. Raichuk, M. Geist, and O. Pietquin. Continuous control with action quantization from demonstrations. In *International Conference on Machine Learning*, pages 4537–4557. PMLR, 2022.

[6] N. Elhami Fard and R. R. Selmic. Adversarial attacks on heterogeneous multi-agent deep reinforcement learning system with time-delayed data transmission. *Journal of Sensor and Actuator Networks*, 11(3):45, 2022.

[7] M. Figura, K. C. Kosaraju, and V. Gupta. Adversarial attacks in consensus-based multi-agent reinforcement learning. In *Proc. of American Control Conference (ACC)*, pages 3050–3055, 2021. doi: 10.23919/ACC50511.2021.9483080.

[8] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell. Adversarial policies: Attacking deep reinforcement learning. In *Proc. of ICLR*, 2019.

[9] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *Proc. of ICLR*, 2015.

[10] J. Guo, Y. Chen, Y. Hao, Z. Yin, Y. Yu, and S. Li. Towards comprehensive testing on the robustness of cooperative multi-agent reinforcement learning. In *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 115–122, 2022.

[11] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Proc. of AAMAS*, pages 66–83. Springer, 2017.

[12] T. Haider, K. Roscher, F. Schmoeller da Roza, and S. Günnemann. Out-of-distribution detection for reinforcement learning agents with probabilistic dynamics models. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 851–859, 2023.

[13] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver. Distributed prioritized experience replay. 2018.

[14] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press, 2012.

[15] K. Kazari, E. Shereen, and G. Dán. Decentralized anomaly detection in cooperative multi-agent reinforcement learning. In *Proc. of IJCAI*, 2023.

[16] K. Kazari, A. Kanellopoulos, and G. Dán. Quickest detection of adversarial attacks against correlated equilibria. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 13961–13968, 2025.

[17] E. L. Lehmann, J. P. Romano, and G. Casella. *Testing statistical hypotheses*, volume 3. Springer, 2005.

[18] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In *Proc. of ICML*, pages 3053–3062, 2018.

[19] J. Lin, K. Dzeparoska, S. Q. Zhang, A. Leon-Garcia, and N. Papernot. On the robustness of cooperative multi-agent reinforcement learning. In *Proc. of IEEE Security and Privacy Workshops (SPW)*, pages 62–68, 2020.

[20] 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.

[21] C. Lischke, T. Liu, J. McCalmon, M. A. Rahman, T. Halabi, and S. Alqahtani. LSTM-based anomalous behavior detection in multi-agent reinforcement learning. In *Proc. of IEEE CSR*, pages 16–21. IEEE, 2022.

[22] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Proc. of NeurIPS*, 2017.

[23] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. LSTM-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[24] K. Mo, W. Tang, J. Li, and X. Yuan. Attacking deep reinforcement learning with decoupled adversarial policy. *IEEE Transactions on Dependable and Secure Computing*, 20(1):758–768, 2022.

[25] M.-h. Oh and G. Iyengar. Sequential anomaly detection using inverse reinforcement learning. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1480–1490, 2019.

[26] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1-2):100–115, 06 1954. ISSN 0006-3444. doi: 10.1093/biomet/41.1-2.100. URL https://doi.org/10.1093/biomet/41.1-2.100.

[27] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proc. of AAMAS*, 2018.

[28] A. Russo and A. Proutiere. Towards optimal attacks on reinforcement learning policies. In *Proc. of American Control Conference (ACC)*, pages 4561–4567, 2021.

[29] A. Sakryukin, C. Raïssi, and M. Kankanhalli. Inferring dqn structure for high-dimensional continuous control. In *International Conference on Machine Learning*, pages 8408–8416. PMLR, 2020.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] A. Sedlmeier, R. Müller, S. Illium, and C. Linnhoff-Popien. Policy entropy for out-of-distribution classification. In *Proc. of International Conference on Artificial Neural Networks*, pages 420–431, 2020.

[32] I. Shames, A. M. Teixeira, H. Sandberg, and K. H. Johansson. Distributed fault detection for interconnected second-order systems. *Automatica*, 47(12):2757–2764, 2011. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2011.09.011. URL https://www.sciencedirect.com/science/article/pii/S0005109811004511.

[33] E. Shereen, K. Kazari, and G. Dán. Adversarial robustness of multi-agent reinforcement learning secondary control of islanded inverter-based AC microgrids. In *Proc. of IEEE SmartGridComm*, pages 1–7, 2023. doi: 10.1109/SmartGridComm57358.2023.10333903.

[34] B. G. Tekgul, S. Wang, S. Marchal, and N. Asokan. Real-time adversarial perturbations against deep reinforcement learning policies: attacks and defenses. In *European Symposium on Research in Computer Security*, pages 384–404. Springer, 2022.

[35] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. In *Proc. of NeurIPS*, pages 15032–15043, 2021.

[36] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black. Revisiting parameter sharing in multi-agent deep reinforcement learning. 2020.

[37] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[38] C. Wang, S. Erfani, T. Alpcan, and C. Leckie. Oil-ad: An anomaly detection framework for sequential decision sequences. *arXiv preprint arXiv:2402.04567*, 2024.

[39] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang. Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection. In *Proc. of ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3726–3734, 2021.

[40] L. Xie, S. Zou, Y. Xie, and V. V. Veeravalli. Sequential (quickest) change detection: Classical results and new directions. *IEEE Journal on Selected Areas in Information Theory*, 2(2):494–514, 2021.

[41] R. Yan, Y. Wang, Y. Xu, and J. Dai. A multiagent quantum deep reinforcement learning method for distributed frequency control of islanded microgrids. *IEEE Transactions on Control of Network Systems*, 9(4):1622–1632, 2022. doi: 10.1109/TCNS.2022.3140702.

[42] Z. Yang and H. Nguyen. Recurrent off-policy baselines for memory-based continuous control. In *Proc. of NeurIPS Deep Reinforcement Learning Workshop*, 2021.

[43] D. Ye, M.-M. Chen, and H.-J. Yang. Distributed adaptive event-

triggered fault-tolerant consensus of multiagent systems with general linear dynamics. *IEEE Trans. on Cybernetics*, 49(3):757–767, 2019.

[44] L. Yu, Y. Qiu, Q. Yao, Y. Shen, X. Zhang, and J. Wang. Robust communicative multi-agent reinforcement learning with active defense. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17575–17582, 2024.

[45] H. Zhang, K. Sun, bo xu, L. Kong, and M. Müller. A distance-based anomaly detection framework for deep reinforcement learning. *Transactions on Machine Learning Research (TMLR)*, Oct 2024.

[46] Y. Zhu, Z. Wang, Y. Zhu, C. Chen, and D. Zhao. Discretizing continuous action space with unimodal probability distributions for on-policy reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Appendix

A Accuracy of the Gaussian Assumption

Proposition 1 states that if $a_t^j|o_t^i$ is Gaussian then the normality score z_t^{ij} has a known and constant expected value. To see why the Gaussian assumption may be reasonable, let us assume that the action of agent j (policy) is determined by o^j and $a^j|o^j \sim N(\mu(o^j), \Sigma)$. Then if $o^j|o^i$ is also Gaussian, then it can be shown that in some scenarios (e.g., when $\mu(o^j)$ is a linear function), $a^j|o^i$ is Gaussian.

To provide an empirical validation, we computed the episodic average of z_t^{ij} across all environments and compared it with its expected value under the Gaussian assumption. Table 7 shows that the empirical average of the normality score under normal agent behavior closely matches its expected value under the Gaussian assumption. This indicates that the proposed Gaussian approximation is a reasonable choice for the considered environments.

We acknowledge that there could be environments where the Gaussian assumption is a poor approximation. For instance, if $o^j|o^i$ is multi-modal (e.g., a mixture model), then $a^j|o^i$ can also follow a multi-modal distribution and a single Gaussian may not approximate the resulting action distribution well. Notably, such a scenario implies that an agent's observations are not informative about other agents' observations, which would hinder distributed anomaly detection in such environments. In such environments a centralized approach might be needed.

B MARL Environments

B.1 Multiwaker

In the *Multiwalker* [11] environment, a set of bipedal robots cooperate to carry a package towards the right side of a terrain. The agents obtain a positive reward based on the change in the position of the package and a large negative reward in case the package falls. Each walker applies force to two joints, one in each leg, resulting in a 4-dimensional action space. The observation of each agent contains information about its legs and joints, as well as sensor measurements of the agent vicinity and neighboring agents. This environment is particularly interesting for our problem, due to the noisy nature of the sensor measurements. We considered a scenario with 5 agents for this benchmark.

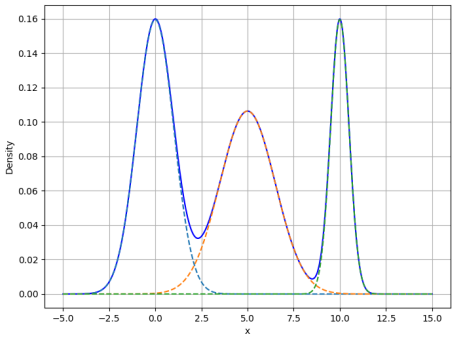


Figure 5. An example of one dimensional multi-modal distribution

	Multiwalker	Tag	World Comm	Pistonball
$\mathbb{E}[z_t^{ij}]$	-2	-2.5	-4.5	-0.5
Average z_t^{ij}	-2.010	-2.490	-4.441	-0.508

Table 7. Episodic average and the expected value of normality score under the Gaussian assumption .

B.2 Tag

Tag (Simple Tag) [22] is one of the Multi-agent Particle Environments (MPE) family. In this environment, a team of three predators try to hunt a single prey. Predators are rewarded based on the number of times they can hit the prey in a limited duration (25 time-steps). Each agent's action is a 5-dimensional vector determining how the agent moves. Each agent observes the other agents' velocities and relative positions as part of the observation vector.

B.3 World Comm

World Comm [22] is another MPE scenario including two competing groups of agents, which we refer to as red and green agents. Green agents' objective is to collect food items and escape from red agents. Red agents are rewarded based on the proximity to green agents. There are forests where agents can hide themselves. We considered a scenario with 4 red agents and one green agent. The red team is trained using c-MARL and the green agent behaves based on a predefined policy. One of the red agents is the leader and is capable of observing all agents at all times and can coordinate its team using a 9-dimensional action vector. Other agents' actions are 5-dimensional.

B.4 Pistonball

Pistonball is one of the Butterfly environments in PettingZoo, built on visual Atari spaces and powered by the Chipmunk physics engine. In this environment, a group of pistons collaborates to move a ball from the right side to the left side of the game border using vertical movements. Each piston's action is a scalar in (-1, 1), representing the amplitude and direction of its vertical movement. The observation for each piston is an RGB image of its surrounding area, including neighboring agents. The reward is determined by the extent of the ball's movement toward the left side. We considered a scenario with 10 agents.

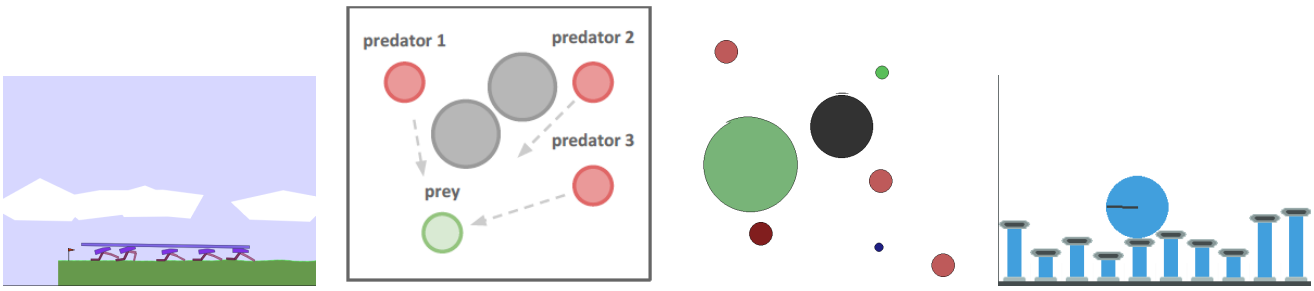
C Other Results

C.1 Effect of hyperparameter w

A hyperparameter for CUSUM detection in (6) is w, which controls the sensitivity of the detection. Table 8 shows the AUC score of PGC with different values of w.

C.2 Time to Detection

One important metric for the performance assessment of a detector is the time to detection. We define this metric as the average number of time steps from the start of the attack until it is detected. The averaging is taken only over true positive episodes, i.e., episodes in which a detection happens are considered. Figure 7 shows the time to detect as a function of false positive rate in all environments. The figure shows that all the impactful attacks (e.g., ACT) can be detected by PGC in less than 5 time steps (or even instantaneously in Multiwalker) already at a very low false positive rate. Another notable observation is that the time to detect of impactful attacks is not significantly affected by their stealthiness, as it takes almost the same time to detect the ACT and the DYN attacks in most of the scenarios.



(a) Multiwalker (b) Tag (c) World Comm (d) Pistonball

Figure 6. Evaluated MARL Environments

	Multiwalker			Tag			World Comm			Pistonball		
Attacks	$w=0$	$w=0.5$	$w=1$	$w=0$	$w=0.75$	$w=1.5$	$w=0$	$w=0.5$	$w=1$	$w=0$	$w=0.05$	$w=0.4$
ACT	0.79	0.99	0.98	0.96	0.99	0.98	0.98	0.99	0.99	0.98	0.99	0.97
RAND	0.85	0.99	0.55	0.85	0.84	0.86	0.56	0.67	0.63	0.99	0.99	0.99
GRAD	0.57	0.67	0.68	0.63	0.65	0.64	0.81	0.88	0.78	0.56	0.58	0.52
DYN1	0.58	0.92	0.91	0.96	0.98	0.98	0.98	0.99	0.98	0.72	0.90	0.8
DYN2	0.70	0.95	0.91	0.94	0.96	0.96	0.91	0.91	0.90	0.92	0.87	0.67

Table 8. AUC score of PGC obtained using different values of w.

C.3 Window-Based Detection

A variation of our proposed detector replaces CUSUM with a window-based averaging of the normality score. Table 9 presents the AUC scores of this scheme against various attacks across all environments. Comparing these results with the AUC scores obtained by PGC (Table 2) reveals that CUSUM significantly enhances detection performance overall.

	Multiwalker	Tag	World Comm	Pistonball
ACT	0.99	0.97	0.97	0.99
RAND	0.68	0.80	0.55	0.83
Grad	0.70	0.57	0.87	0.61
DYN1	0.96	0.97	0.97	0.90
DYN2	0.89	0.95	0.97	0.81

Table 9. AUC score of the proposed detector with window-based detection.

C.4 Observation Tracking Baseline

As another baseline for comparison, we evaluated an anomaly detection approach based on the sequence of observations. Specifically, we adapted the detection method proposed by [45] to the multi-agent setting, where each agent independently tracks its own observation sequence to detect potential anomalies. This approach is justified by the fact that the victim agent's actions can cause deviations in the observation trajectories of non-victim agents, making it possible for them to detect anomalies affecting the c-MARL team. However, unlike our method, this approach cannot pinpoint the exact source of these abnormalities, i.e., the victim agent.

In this method, which we refer to as *Observation Tracking*, for each action type, the mapped feature vectors of observations leading to that action are fitted to a multivariate Gaussian distribution. During execution, the minimum Mahalanobis distance between a new observation and any of the Gaussian models is used as the normality score. If this score exceeds a predefined threshold, an anomaly is detected. Since this method is designed for discrete action setups, we applied the same quantization technique as in *Discrete* with $N_q = 3$. For feature selection, we followed the approach in [45], using 30 components for PCA transformation.

	Multiwalker	Tag
ACT	0.61	0.55
RAND	0.5	0.5
Grad	0.5	0.5
DYN1	0.55	0.52
DYN2	0.52	0.51

Table 10. AUC score of the observation tracking approach.

Table 10 presents the AUC scores of the Observation Tracking method for detecting various attacks in Multiwalker and Tag. The results highlight the limitations of this approach in distributed detection of attacks on agents' actions, emphasizing the necessity of directly tracking actions rather than relying solely on observation sequences.

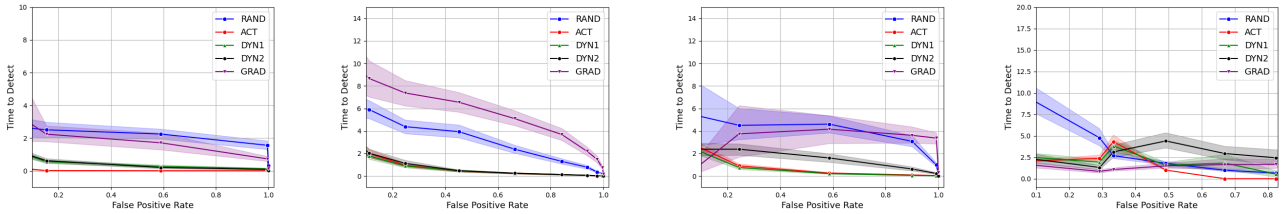
D Hyperparameters

D.1 Training Detectors and Attacks

The hyperparameters for training detectors are shown in table 11. The value of λ used for training the DYN1 attack in Multiwalker, Tag, World Comm, and Pistonball, were 5, 1, 0.5, and 1, respectively. For DYN2 attack these values were 10, 2, 5, and 2, respectively.

D.2 Training c-MARL

The parameters we used for training c-MARL policies are summarized in Table 12.



(a) Multiwalker (b) Tag (c) World Comm (d) Pistonball

Figure 7. Time to detect vs. false positive rate for the proposed method against different attacks

Hyperparameter	Value
Use_rnn	True
hidden_size (Pistonball)	256
hidden_size (other environments)	128
batch_size	20
include_last_act(Multiwalker)	True
include_last_act(Tag/World Comm)	False
lr	5e-4

Table 11. Hyperparameters used for training predictors.

Environment	Hyperparameter	Value
Multiwalker	Algorithm	PPO
	gamma	0.99
	kl_coeff	0.2
	kl_target	0.01
	lambda	1
	lr	5e-5
	use_lstm	True
	lstm_cell_size	128
	max_seq_len	25
	post_fcnet_activation	relu
	vf_share_layers	False
	sgd_minibatch_size	128
Tag/World Comm	Algorithm	APE-X DDPG
	gamma	0.99
	clip_actions	False
	actor_hiddens	[400, 300]
	critic_hiddens	[400, 300]
	lr	0.0005
	adam_epsilon	1e-8
	train_batch_size	512
	use_lstm	False
	vf_share_layers	True
	compress_observations	False
	post_fcnet_activation	relu
Pistonball	Algorithm	PPO
	gamma	0.99
	kl_coeff	0.2
	kl_target	0.01
	lambda	1
	lr	5e-5
	use_lstm	False
	post_fcnet_activation	relu
	vf_share_layers	False
	sgd_minibatch_size	128

Table 12. Hyperparameters used for training c-MARL.