TooBadRL: Trigger Optimization to Boost Effectiveness of Backdoor Attacks on Deep Reinforcement Learning

Songze Li[*1], Mingxuan Zhang[*1], Kang Wei[✉1], Shouling Ji[2]
[1]Southeast University, China
[2]Zhejiang University, China
{songzeli, mingxuan.zhang, kang.wei}@seu.edu.cn, {sji}@zju.edu.cn

***Abstract*—Deep reinforcement learning (DRL) has achieved remarkable success in a wide range of sequential decision-making domains, including robotics, healthcare, smart grids, and finance. Recent research demonstrates that attackers can efficiently exploit system vulnerabilities during the training phase to execute backdoor attacks, producing malicious actions when specific trigger patterns are present in the state observations. However, most existing backdoor attacks rely primarily on simplistic and heuristic trigger configurations, overlooking the potential efficacy of *trigger optimization*. To address this gap, we introduce `TooBadRL` (Trigger Optimization to Boost Effectiveness of Backdoor Attacks on DRL), the first framework to systematically optimize DRL backdoor triggers along three critical axes, i.e., temporal, spatial, and magnitude. Specifically, we first introduce a performance-aware adaptive freezing mechanism for injection timing. Then, we formulate dimension selection as a cooperative game, utilizing Shapley value analysis to identify the most influential state variable for the injection dimension. Furthermore, we propose a gradient-based adversarial procedure to optimize the injection magnitude under environment constraints. Evaluations on three mainstream DRL algorithms and nine benchmark tasks show that `TooBadRL` significantly improves attack success rates, while ensuring minimal degradation of normal task performance. These results highlight the previously underappreciated importance of principled trigger optimization in DRL backdoor attacks. The source code of `TooBadRL` can be found at https://github.com/S3IC-Lab/TooBadRL.**

1. Introduction

Deep Reinforcement Learning (DRL) can learn complex, high-dimensional tasks. It is an important method for sequential decision-making. DRL has led to progress in areas such as large language models [1], healthcare [2], energy management [3], autonomous vehicles [4], and finance [5]. DRL succeeds by combining the ability of deep neural networks (DNNs) to represent data with the trial-and-error learning process of Reinforcement Learning (RL) [6].

As DRL systems are increasingly deployed in safety-critical and mission-sensitive scenarios, concerns about their security and reliability have come to the forefront [7], [8], [9], [10], [11]. Among various threats, backdoor attacks [12], [13], [14] have emerged as a particularly insidious risk. In such attacks, an attacker covertly implants a hidden trigger during the training phase, producing malicious actions when specific trigger patterns are present in the state observations, which can typically be divided into two steps. The first step is the trigger injection, in which the attacker injects a predefined trigger pattern into a subset of observed states. The second step is the reward manipulation, in which the attacker alters reward signals to incentivize a malicious action or by directly forcing the agent to execute it, thereby forging a strong association between the trigger's presence and the desired deviant behavior. These attacks can potentially undermine even the most powerful DRL agents, eroding trust in automated decision-making systems [15].

While existing studies have established foundational methodologies for DRL backdoor attacks, e.g., [16], [17], [18], exploring the system vulnerabilities partly. However, the vast majority of existing methods rely on overly simple, heuristic trigger configurations. Most notably, triggers are often injected by assigning fixed boundary or midpoint values to arbitrarily or randomly chosen state dimensions, without any principled selection or data-driven optimization [19], [20], [21]. As a result, these attacks often have limited effectiveness or cause significant degradation of the agent's normal task performance, thus increasing the likelihood of their detection and failing to investigate the underlying vulnerabilities.

In fact, the design of a trigger itself is comprising *temporal* considerations (when to inject the trigger), *spatial* selection (which dimension to perturb), and *magnitude* assignment (what value to assign), which is the cornerstone determining the success or failure of a DRL backdoor attack. Triggers poorly aligned with the agent's internal representation, improperly valued, or injected prematurely often either fail to activate reliably or severely degrade the agent's benign performance. The trigger in existing works is implemented using a fixed value, with no principled dimension selection [13], [14], [22]. To investigate the impact of trigger design, we systematically vary both the trigger dimension and value, and compare their performance. The

[*] *Songze Li and Mingxuan Zhang contributed equally to this work.*
[✉] *Correspondence to Kang Wei.*

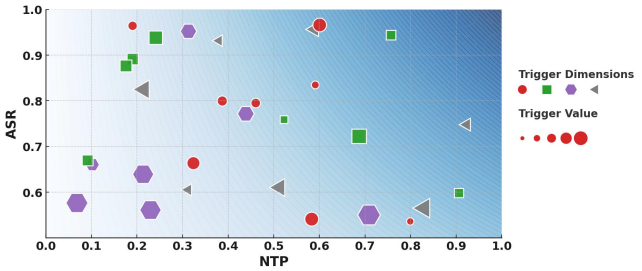

Figure 1: Impact of trigger dimension (marker color) and value (marker size) on attack and benign performance. A darker background (top right corner) denotes desirable regions, where a high attack success rate is achieved without compromising normal task performance.

results, as depicted in Figure 1, reveal two key findings: (1) Both attack efficacy and normal task performance show significant fluctuations, and in some cases, they may even experience complete failure, which depends on the specific trigger configuration; (2) Unoptimized and non-selective triggers prove to be unreliable and ineffective in practice.

To overcome these fundamental limitations, this paper introduces TooBadRL (Trigger Optimization to Boost Effectiveness of Backdoor Attacks on DRL), a unified framework for systematic trigger optimization in DRL backdoor attacks. To the best of our knowledge, TooBadRL is the first to comprehensively optimize backdoor trigger design along three critical axes: *temporal* (optimal injection timing), *spatial* (selection of state dimension), and *magnitude* (value assignment), resulting in robust and highly effective attacks with minimal impact on benign performance.

Specifically, TooBadRL integrates three core methodological innovations: (1) **Performance-Aware Adaptive Freezing Mechanism.** To minimize interference with normal policy learning and determine the optimal *temporal* juncture for attack initiation, we propose an adaptive freezing mechanism. This mechanism employs statistical hypothesis testing to introduce triggers only after verifying that the agent's main-task policy has stabilized, thereby preventing premature policy degradation and ensuring the backdoor implant remains robust. (2) **Trigger Dimension Selection via Shapley Values.** For *spatial* optimization, we frame the selection of the optimal trigger dimension as a cooperative game among state dimensions. Employing Shapley value analysis, we quantify each dimension's contribution to policy outputs. This principled, game-theoretic approach systematically identifies the dimension whose perturbation most significantly influences the agent's policy, ensuring maximal attack effectiveness. (3) **Gradient-Based Adversarial Magnitude Optimization.** Once the trigger dimension is selected, TooBadRL applies a gradient-based adversarial optimization method to determine the optimal trigger *magnitude*. Leveraging the differentiability of DRL policies, we iteratively adjust the trigger's value to maximize the agent's probability of selecting the attacker-specified target action (in discrete action spaces) or minimize the distance to a desired target action vector (in continuous action spaces), crucially respecting environment-defined bounds to ensure subtlety.

We evaluate TooBadRL comprehensively through experiments spanning three mainstream DRL algorithms, i.e., Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO), Advantage Actor-Critic (A2C), across nine diverse environments, comparing against five attack baselines. These extensive experimental results validate the substantial effectiveness of our principled trigger optimization approach as follows. Firstly, TooBadRL achieves exceptionally high attack success rates; for instance, with PPO, attack success rates reach up to 0.9994 on Acrobot, 0.9796 on BipedalWalker, 0.9711 on Pendulum, and 0.9833 on Hopper. Crucially, these high attack success rates are achieved without compromising normal task performance, which remains near the level of clean agents. Secondly, detailed ablation studies confirm the necessity of each optimization component; heuristic configurations lead to significantly lower attack success rates and greater performance trade-offs, affirming the value of our systematic approach. Finally, we demonstrate robustness against existing defense methods, with experiments indicating that our optimized triggers resist common detection and elimination strategies, thereby empirically validating the attack's operational feasibility in practical deployment scenarios.

In summary, our contributions are as follows:

- We propose TooBadRL, the first principled framework to design backdoor triggers on DRL, which simultaneously optimizes trigger injection timing, dimension selection, and value assignment. The propose TooBadRL significantly enhances attack effectiveness while preserving benign task performance.
- Through extensive empirical evaluations across nine benchmark environments and three leading DRL algorithms (PPO, TRPO and A2C), we demonstrate that TooBadRL achieves state-of-the-art attack success rates, substantially outperforming existing methods.
- We conduct comprehensive ablation studies and robustness analyses against contemporary defense mechanisms. These investigations not only validate the individual importance of each component within TooBadRL but also expose the vulnerabilities of current DRL systems to sophisticated, optimized backdoor threats.

2. Related Work

Recently, backdoor attacks have attracted a lot of attention in DNN empowered domains: such as computer vision [23], [24], [25], natural language processing [26], [27], [28], malware detection [29], [30], graph neural networks [31], [32], [33], self-supervised learning [34], [35] and distributed learning [36], [37], [38]. Meanwhile, backdoor attacks present a significant threat to DRL [39], [40], [41], [42].

Research on backdoor attacks in DRL has investigated vulnerabilities across various learning phases. For instance, some attacks focus on the offline learning phase, where agents learn from a fixed dataset. A notable example is BAFFLE, which poisons a small fraction of the offline dataset

to significantly degrade the agent's performance [19]. In contrast, other research focuses on online learning phase, where the agent learns from continuous interaction. In this area, BACKDOORL introduces a backdoor for competitive RL that forces a victim to fail [21], while other works have designed temporal-pattern triggers for partially observable states [43] and MARNet has targeted cooperative multi-agent systems [20].

While these attacks operate in different settings, a more fundamental distinction lies in the sophistication of their trigger design. We can classify them into two main groups: (1) Static and Heuristic Trigger Attacks. This dominant category includes attacks that use triggers defined by fixed rules, simple heuristics, or pre-selected patterns. The trigger design is often manual and not systematically optimized against the specific agent. For example, the seminal work TrojDRL typically uses fixed values on pre-selected state dimensions [13]. Similarly, while a more recent framework like UNIDOOR adaptively tunes its reward function, its trigger mechanism remains a simple, non-optimized threshold value. [16]. The simplicity of these static triggers limits their effectiveness and stealth. (2) Adaptive and Optimized Trigger Attacks. These methods leverage training information about the DRL agent to optimize triggers. For example, BadRL takes a step in this direction by using mutual information to help select which state dimensions to perturb [14]. However, existing approaches still tend to focus on optimizing only a single aspect of the trigger (e.g., the spatial dimension) in isolation.

To the best of our knowledge, no existing work has holistically optimized the backdoor trigger across all its critical facets. This leaves a significant gap, as uncoordinated trigger design can lead to suboptimal attacks. Our work, `TooBadRL`, directly addresses this limitation. We propose the first framework to systematically and jointly optimize DRL backdoor triggers along three critical axes: *temporal* (when to inject), *spatial* (which dimension to perturb), and *magnitude* (what value to use).

3. Backdoor Attack on Deep Reinforcement Learning

This section formally defines the problem of designing and optimizing backdoor attacks against DRL agents. Specifically, we provide the necessary background on DRL, describe the characteristics of backdoor attacks within this context, and then formulate the mathematical objectives for achieving an effective attack.

3.1. Deep Reinforcement Learning

DRL is typically modeled as a Markov Decision Process (MDP) [44], defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, T, \gamma)$. This includes the state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R(s_t, a_t)$, state-transition probability function $T(s_{t+1}|s_t, a_t)$, and discount factor $\gamma \in [0, 1)$.

At each timestep t, the agent observes state $s_t \in \mathcal{S}$ and selects action $a_t \in \mathcal{A}$ according to its policy $\pi_\theta(a_t|s_t)$, often parameterized by a DNN with weights θ. The environment then provides a reward r_t and transitions to a new state s_{t+1}. This interaction generates a trajectory $\tau = \{(s_0, a_0, r_0), \ldots, (s_H, a_H, r_H)\}$. The agent's objective is to learn optimal policy parameters θ^* that maximize the expected cumulative discounted reward:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{H} \gamma^t r_t\right]. \tag{1}$$

DRL uses DNNs to approximate π_θ and/or value functions (e.g., $V_\pi(s)$, $Q_\pi(s, a)$), making it suitable for complex inputs like images or sensor data [45], [46], [47]. Common DRL algorithms include PPO [48], TRPO [49], and A2C [50].

3.2. Threat Model

Attacker's Objective. We consider an attacker aiming to surreptitiously compromise a DRL agent by implanting a hidden backdoor. This backdoor compels the agent to execute attacker-specified actions when a trigger is embedded in its observations, while ensuring normal behavior and normal task performance in the trigger's absence. In addition, it should be hard to find by human checks or standard defenses. This means it should only slightly change normal state observations.

Attacker's Capabilities and Knowledge. We assume an attacker knows the target DRL agent's environment well, in which it might compromise a data source or partly influence the training environment without full access to how the agent is developed and used. This includes its state space $\mathcal{S}$, action space $\mathcal{A}$, and how states change T. The attacker can inject triggers into states at the chosen time. As we know, if the attacker can see the agent's learning progress, it will optimize trigger placement. However, these attackers do not directly control or know the details of the DRL training algorithm (like learning rates or network structure) or the final learned parameters θ^* of a normal agent. This threat model shows realistic situations.

3.3. Problem Formulation

The core of backdoor attacks against DRL systems during training phase lies in forging a strong association between the presence of the trigger and the desired deviant behavior. If a D-dimensional state s is injected in dimensions $\mathcal{I}_{\text{trig}} \subseteq \{1, \ldots, D\}$ with values $v_{\text{trig},j}$, the triggered state s' becomes:

$$s'_j = \begin{cases} v_{\text{trig},j}, & \text{if } j \in \mathcal{I}_{\text{trig}} \\ s_j, & \text{otherwise.} \end{cases} \tag{2}$$

The attacker injects these triggered states during training and manipulates learning signals to link the trigger to a_{target}.

Many existing backdoor attacks in DRL use simple, unoptimized triggers (e.g., random dimensions, fixed values) [16], [19]. This often leads to poor attack success or significant

harm to the agent's normal task performance, making detection easier. Our work addresses this by systematically optimizing the trigger design.

Formally, a normal DRL agent learns a policy π_{θ^*} in an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, T, \gamma)$ to maximize expected rewards (see Section 3.1 and Equation 1). The attacker aims to train a compromised policy $\pi_{\theta'}$ using a trigger function $\delta : \mathcal{S} \to \mathcal{S}$ (where $s' = \delta(s)$ as in Eq. 2). This policy must satisfy two objectives:

- **Attack Effectiveness:** Maximize the probability that $\pi_{\theta'}$ selects a_{target} when given a triggered state $s' = \delta(s)$:

$$\max_{\theta', \delta} \mathbb{E}_{s \sim \mathcal{D}_{\mathcal{S}}} \left[P(\pi_{\theta'}(a|s') = a_{\text{target}}) \right]. \tag{3}$$

- **Normal Performance Preservation:** Ensure the performance of $\pi_{\theta'}$ on normal states is close to that of π_{θ^*}, keeping the difference in state-value functions bounded by ϵ:

$$\mathbb{E}_{s \sim \mathcal{D}_{\mathcal{S}}} \left[|V^{\pi_{\theta'}}(s) - V^{\pi_{\theta^*}}(s)| \right] \leq \epsilon. \tag{4}$$

The problem is to devise an attack strategy (designing δ and training $\pi_{\theta'}$) that solves:

$$\begin{aligned} \max_{\theta', \delta} \quad & \mathbb{E}_{s \sim \mathcal{D}_{\mathcal{S}}} \left[P(\pi_{\theta'}(a|\delta(s)) = a_{\text{target}}) \right] \\ \text{subject to} \quad & \mathbb{E}_{s \sim \mathcal{D}_{\mathcal{S}}} \left[|V^{\pi_{\theta'}}(s) - V^{\pi_{\theta^*}}(s)| \right] \leq \epsilon. \end{aligned} \tag{5}$$

In such attack, when this trigger is absent, the DRL agent acts normally. When the trigger appears in the agent's observed state, it causes the agent to perform a pre-defined malicious action, $a_{\text{target}} \in \mathcal{A}$.

Our framework, `TooBadRL`, focuses on finding an effective trigger δ through optimization to better satisfy these conditions. A well-optimized trigger is key to achieving high attack effectiveness (Eq. 3) while preserving normal performance (Eq. 4), as it can induce the malicious action more reliably with minimal state changes.

4. Trigger Optimization of `TooBadRL`

A DRL backdoor attack generally involves two main stages. The first is the *attack preparation stage*, where the attacker designs and optimizes the backdoor trigger. The second is the *backdoor injection stage*, where the attacker uses this trigger to compromise the DRL agent during its training process. Our work, `TooBadRL`, focuses on the first stage: systematic trigger optimization. As outlined in our Problem Statement (Section 3.3), an effective backdoor attack needs a carefully designed trigger δ. This trigger should reliably cause the target malicious action a_{target} when present, without significantly harming the agent's normal task performance when absent. The characteristics of the trigger can be summarized as follows: *when* it is introduced, *which* state dimension it perturbs, and *what* value it assigns. This section details the core of `TooBadRL`: a principled approach to optimize these three trigger aspects. By optimizing the trigger in the preparation stage, we aim to provide a highly effective trigger that can then be used in the subsequent backdoor injection stage (discussed in Chapter 5) to achieve the attack objectives defined in Equation 5. A well-optimized trigger directly improves the attack's ability to meet these objectives by increasing the probability of the target action upon activation while minimizing interference with normal operations.

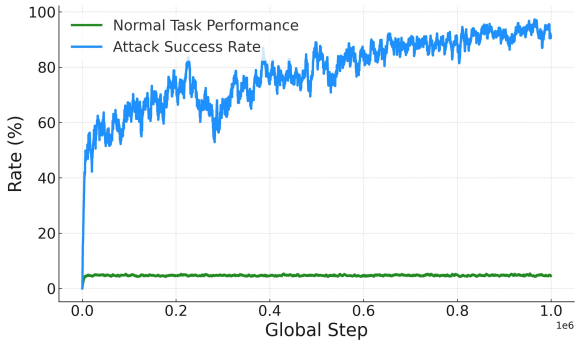

Figure 2: The evolution of normal task performance and attack success rate for the Hopper task when backdoor attacks are introduced from the start of training without a freeze period. Early attack injection rapidly increases attack success rate but causes persistent suppression of normal task performance (normal task performance remains near zero throughout training).

4.1. When – Adaptive Freezing Mechanism

When to inject backdoor triggers is critical, which helps balance attack performance with keeping normal task performance. Figure 2 shows that injecting triggers too early can stop the agent from learning the normal task. This operation often leads to policies that are always unstable or perform badly as recognized in [16]. Although existing study has already adopted a *freeze period*, where the agent trains only on the normal task without backdoor interference. However, these methods usually use a fixed, predetermined time. This fixed method cannot adapt to different environments, agent learning speeds, or task difficulties.

To solve this, we suggest a new adaptive freezing mechanism. It dynamically finds the right time to end the freeze period based on how the agent is learning. During this first freeze period, the agent trains only to master the normal task. Our mechanism finds the point when the agent's improvement in normal task performance slows down. This indicates that the policy has largely converged and is robust enough to withstand backdoor triggers with minimal disruption.

This adaptive mechanism works by regularly checking the agent's normal task performance every T_{eval} training steps. Let P_i be the average total reward during the i-th check, which shows normal task performance. We keep a history of these performance scores, $\mathcal{P} = \{P_1, P_2, \ldots, P_t\}$. We use two separate sliding windows, each of size k: a previous window $W_{\text{prev}} = \{P_{t-2k+1}, \ldots, P_{t-k}\}$ and a current window $W_{\text{curr}} = \{P_{t-k+1}, \ldots, P_t\}$. To check if the agent's performance has stabilized, we use the Wilcoxon Signed-rank Test [51]. This test compares the performance

scores in W_{prev} and W_{curr}. The Wilcoxon test is a non-parametric test for paired samples. It is good here because it does not assume anything about how the performance data is distributed. This test gives a p-value, p_{value}. If $p_{\text{value}} > \alpha$, where α is a set significance level (e.g., 0.05), we decide that the performance improvements in W_{curr} are not statistically significant compared to W_{prev}. At this point, the freeze period ends. The agent is considered to have a stable policy for the normal task. Algorithm 2 (shown in Appendix A) gives more details on this adaptive freezing mechanism.

4.2. Which – Trigger Dimension Selection

After the adaptive freezing mechanism stabilizes the agent policy (Section 4.1), our method optimizes the backdoor trigger *spatially*. This means finding the most influential dimensions in the agent's D-dimensional state $s = [s_1, \ldots, s_D]$ to perturb. The goal is to select a small subset of state dimensions, $\mathcal{I}_{\text{trig}} \subseteq \{1, \ldots, D\}$, such that changes to these dimensions cause the most significant and predictable alteration in the agent's policy output $f(s)$, maximizing the backdoor's effectiveness.

Traditional feature selection methods often prove inadequate in DRL due to context-dependent feature impacts and temporal correlations [52], [53]. In this way, we utilize Shapley Additive Explanations (SHAP) [54], a model-agnostic framework from cooperative game theory, to quantify each dimension's contribution to the policy output. We define $\phi_j(f, s)$ as SHAP value, which represents the marginal contribution of dimension s_j to $f(s)$. Since exact SHAP computation is often intractable for complex DRL policies, we estimate $\phi_j(f, s)$ using a common approximation technique. This involves fitting a local, weighted linear surrogate model $g(z')$ to the policy's outputs for perturbed state instances:

$$g(z') = \phi_0(s) + \sum_{j=1}^{D} \phi_j(s) z'_j. \tag{6}$$

where z'_j is a binary vector representing feature coalitions, and $\phi_j(s)$ is the estimated SHAP value. The injected state $s_{z'}$ is constructed by replacing features absent from a coalition z' with baseline values $v_{\text{bg},k}$ (e.g., means from a background dataset X_{bg}):

$$(s_{z'})_k = \begin{cases} s_k & \text{if } z'_k = 1, \\ v_{\text{bg},k} & \text{if } z'_k = 0. \end{cases} \tag{7}$$

The model $g(z')$ is trained by minimizing a weighted squared loss $\mathcal{L}$, expressed as

$$\mathcal{L}(\phi_0, \ldots, \phi_D) = \sum_{z' \in Z} \left[f(s_{z'}) - g(z') \right]^2 \pi(z'), \tag{8}$$

where weights $\pi(z')$ are given by the Shapley kernel, expressed as

$$\pi(z') = \frac{D-1}{\binom{D}{k} k (D-k)}. \tag{9}$$

Algorithm 1: Trigger Dimension Selection

Input: Agent policy f; Background dataset X_{bg}; Set of evaluation states X_{exp}; Number of trigger dimensions K; Number of SHAP samples per state M_{shap}

Output: Selected trigger dimensions $\mathcal{I}_{\text{trig}}$

```
Compute feature-wise means v_{bg,j} for all j ∈ {1,...,D} using X_bg;
Initialize an empty list for aggregated SHAP values for each dimension;
foreach state s^(i) ∈ X_exp do
    Estimate SHAP values φ_j(f, s^(i)) for all dimensions j by::;
        Generate M_shap coalition vectors z' ∈ {0,1}^D;
        For each z', construct perturbed state s^(i)_{z'} as per Eq. (7);
        Compute policy outputs f(s^(i)_{z'});
        Fit the local linear model g(z') using Shapley kernel weights π(z') and minimizing loss L;
        Extract φ_j(f, s^(i)) for all j;
    Store these φ_j(f, s^(i)) values;
For each dimension j, calculate its global importance I_j using Eq. (10) over all s^(i);
Select the top-K dimensions with the highest I_j values to form I_trig;
return I_trig;
```

Once SHAP values $\phi_j(f, s^{(i)})$ are estimated for all dimensions j across a set of N_{exp} evaluation states $s^{(i)} \in X_{\text{exp}}$, we calculate the global importance I_j for each dimension by averaging its absolute SHAP values:

$$I_j = \frac{1}{N_{\text{exp}}} \sum_{i=1}^{N_{\text{exp}}} \left| \phi_j(f, s^{(i)}) \right|. \tag{10}$$

The state dimensions are ranked by I_j, and the top-K dimensions with the highest global importance are selected as the trigger dimensions $\mathcal{I}_{\text{trig}}$. The detailed steps are shown in Algorithm 1. This SHAP-based selection robustly identifies state features whose manipulation is most likely to alter the agent's behavior effectively. Further implementation details are provided in Appendix B.

4.3. What – Trigger Value Optimization

After selecting the trigger dimension(s) $\mathcal{I}_{\text{trig}}$ (or a single dimension p) (Section 4.2), we need to optimize the trigger's *magnitude*. The goal of trigger value optimization is to find an optimal trigger value v^* for the chosen dimension p. The value v^* should maximally induce the agent's policy π_θ to select the attacker's target action a_{target} when injected into the state. For discrete actions, we can maximize $P(a_{\text{target}})$; for

continuous actions, we can minimize the distance between the output action and $\mathbf{a}_{\text{target}}$.

Instead of using heuristic values, we formulate this as a constrained optimization problem. We address this problem through a gradient-based search, which harnesses the differentiability of DRL policies to enable precise tuning. The detailed steps are as follows.

Optimization Formulation. Given a valid range $[v_{\min}, v_{\max}]$ for dimension p, we find v^* by solving the following optimization problems. For discrete action spaces:

$$v^* = \underset{v \in [v_{\min}, v_{\max}]}{\arg\max}\ \pi_\theta\left(a_{\text{target}} \mid \mathbf{s}_{\text{base}}[p \leftarrow v]\right). \tag{11}$$

For continuous action spaces:

$$v^* = \underset{v \in [v_{\min}, v_{\max}]}{\arg\min}\ \left\|\boldsymbol{\mu}_\theta\left(\mathbf{s}_{\text{base}}[p \leftarrow v]\right) - \mathbf{a}_{\text{target}}\right\|_2^2. \tag{12}$$

In these two formulations, i.e., (11) and (12), $\mathbf{s}_{\text{base}}[p \leftarrow v]$ represents a base state $\mathbf{s}_{\text{base}}$ (e.g., an average state) with its p-th dimension replaced by the candidate trigger value v, and $\boldsymbol{\mu}_\theta(\cdot)$ is the mean action vector predicted by the policy network π_θ.

Gradient-Based Iterative Search. The search begins with an initial value v_0, often the midpoint of the valid range $[v_{\min}, v_{\max}]$. At each iteration k, a candidate state $\mathbf{s}_k$ is formed by substituting the current trigger value v_k into the p-th dimension of the base state $\mathbf{s}_{\text{base}}$. A loss L_k is then computed to quantify how effectively v_k induces the target action a_{target}. For discrete action spaces, this loss is $L_k = -\log(\pi_\theta(a_{\text{target}}|\mathbf{s}_k) + \epsilon_{\log})$, with $\epsilon_{\log}$ for numerical stability. For continuous action spaces, the loss is $L_k = \|\boldsymbol{\mu}_\theta(\mathbf{s}_k) - \mathbf{a}_{\text{target}}\|_2^2$. The gradient of this loss with respect to the trigger value, $g_k = \frac{\partial L_k}{\partial v_k}$, is calculated using backpropagation. The trigger value is then updated for the next iteration, v_{k+1}, using a momentum-based approach: an intermediate update $u_{k+1} = \beta_{\text{mom}} u_k - \eta_k g_k$ is computed, and the new trigger value becomes $v_{k+1} = \text{clip}(v_k + u_{k+1},\, v_{\min},\, v_{\max})$. Here, η_k is the learning rate and β_{mom} is the momentum factor; the clip$(\cdot)$ function ensures v_{k+1} remains within the predefined valid range. This iterative process is repeated for a fixed number of N_{opt} steps or until the change in v_k falls below a convergence threshold, yielding the optimized trigger value v^*.

This gradient-based iterative search method is effective to find a trigger value that reliably elicits the attacker's desired policy response, while adhering to the environment's observation space limits. By carefully optimizing the trigger value, `TooBadRL` maximizes the backdoor's effectiveness and precision, simultaneously minimizing its potential to disrupt the agent's normal task performance. Please see detailed algorithm steps and hyperparameter settings in Appendix C.

5. The Proposed `TooBadRL` Attack

5.1. Overview of `TooBadRL`

After optimizing the backdoor trigger from temporal, spatial, and magnitude aspects (Sections 4.1, 4.2, and 4.3), the final stage of `TooBadRL` is to embed optimized backdoor mechanism into the training process of DRL agent. It is an active process with several key parts: precise trigger injection, dynamic adjustment of attack frequency, strategic action poisoning, and targeted reward manipulation. These parts work together to effectively implant the backdoor while maintaining the agent's normal task performance, in line with the objective in Equation 5. Figure 3 illustrates the framework of TooBadRL.

5.2. Trigger Injection into State Observations

At each timestep t when the agent interacts with the environment, it receives a state observation s_t. If the current timestep is chosen for an attack (decided by the dynamic attack frequency mechanism in Section 5.3), the system injects the optimized trigger. Let the optimized trigger be defined by the chosen dimensions $\mathcal{I}_{\text{trig}}$ and the corresponding optimized values $\mathbf{v}^*_{\text{trig}} = (v^*_j \text{ for } j \in \mathcal{I}_{\text{trig}})$. The trigger injection operation, $\text{Inject}(s_t, \mathcal{I}_{\text{trig}}, \mathbf{v}^*_{\text{trig}})$, changes s_t to a triggered state s'_t. Specifically, for each dimension $j \in \mathcal{I}_{\text{trig}}$, the original state part $s_t[j]$ is replaced with its optimized trigger value $v^*_j \in \mathbf{v}^*_{\text{trig}}$. Dimensions not in $\mathcal{I}_{\text{trig}}$ are not changed. This changed state s'_t then replaces the original state s_t for all calculations in that timestep. This includes the input to the agent's policy network $\pi_\theta(a|s'_t)$ and any related value networks (e.g., $V_\theta(s'_t)$). This ensures that the agent's decision-making and learning updates use the backdoor trigger whenever an attack is active. If the current timestep is not for an attack, s_t is used without change.

5.3. Dynamic Attack Frequency Adaptation

`TooBadRL` adaptively adjusts the trigger injection frequency to balance attack success rate and normal task performance. This mechanism periodically compares current attack success rate (ASR_{curr}) and normal task performance (NTP_{curr}) against their targets (e.g., $\text{ASR}_{\text{target}} = 1.0$, $\text{NTP}_{\text{target}} = 1.0$). The strategy prioritizes achieving the target attack success rate, then refines frequency to minimize normal task performance degradation.

The attack interval, SI_{attack} (steps between trigger injections), is adjusted based on two primary conditions:

Critically Deficient Attack Success Rate: If ASR_{curr} is too low (e.g., $\text{ASR}_{\text{curr}} < 0.5 \times \text{ASR}_{\text{target}}$), the attack frequency is intensified to expedite attack success rate improvement. SI_{attack} is decreased proportionally to the attack success rate deficit:

$$SI_{\text{attack}} \leftarrow \frac{SI_{\text{attack}}}{1 + (\text{ASR}_{\text{target}} - \text{ASR}_{\text{curr}})}. \tag{13}$$

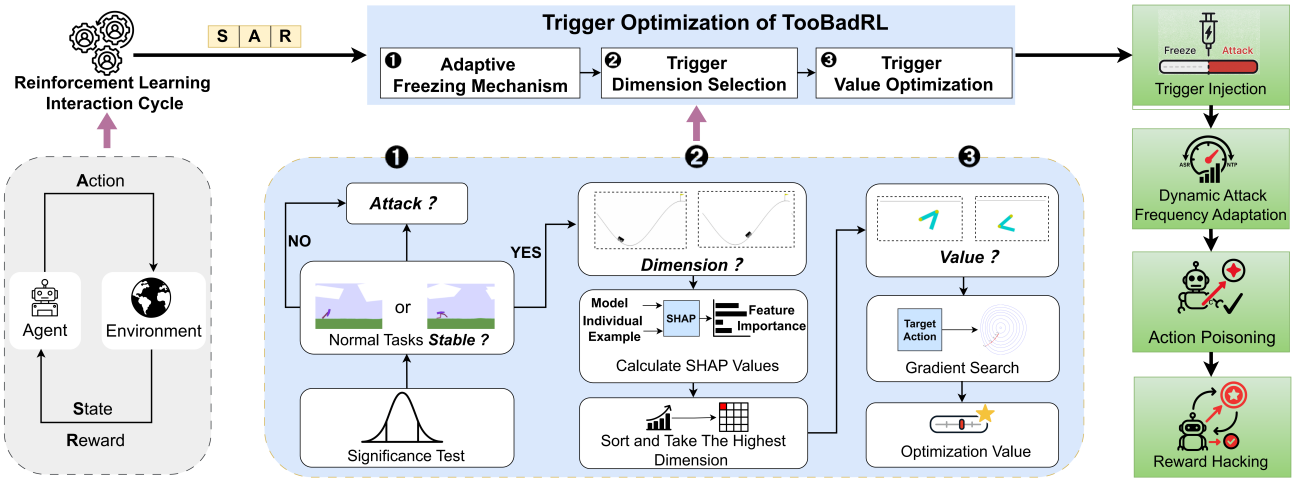

Figure 3: The framework of TooBadRL.

Compromised Normal Task Performance (with Non-Critical Attack Success Rate): If attack success rate is not critically deficient but NTP_{curr} is too low (e.g., $\text{NTP}_{\text{curr}} < 0.5 \times \text{NTP}_{\text{target}}$), the attack frequency is reduced to allow normal task performance to recover. SI_{attack} is increased proportionally to the normal task performance shortfall:

$$SI_{\text{attack}} \leftarrow SI_{\text{attack}} \times (1 + [\text{NTP}_{\text{target}} - \text{NTP}_{\text{curr}}]). \quad (14)$$

If neither condition is met, SI_{attack} remains unchanged. Updates are typically made for tangible changes to prevent excessive oscillation. This adaptive strategy ensures TooBadRL aggressively pursues attack efficacy while being mindful of normal task performance.

5.4. Action Poisoning for Behavior Reinforcement

To make the link between the trigger and the desired bad behavior stronger, TooBadRL includes an action poisoning mechanism. When an attack is active ($f_{\text{attack}} = \text{true}$) and a periodic tampering condition is met (e.g., a tampering counter N_{tamper} satisfies $N_{\text{tamper}} \bmod P_{\text{tamper}} = 0$, where P_{tamper} is a set period), the action a_t chosen by the agent's policy $\pi_\theta(s'_t)$ based on the triggered state s'_t is overridden. It is replaced by the attacker-chosen target action a_{target}:

$$a'_t = \begin{cases} a_{\text{target}} & \text{if } f_{\text{attack}} = \text{true and} \\ & \text{tampering condition met} \\ \pi_\theta(a_t|s'_t) & \text{if } f_{\text{attack}} = \text{true and} \\ & \text{tampering condition not met} \\ \pi_\theta(a_t|s_t) & \text{if } f_{\text{attack}} = \text{false} \end{cases} \quad (15)$$

This changed action a'_t (or the original action if not poisoned) is then performed in the environment. The resulting transition $(s'_t, a'_t, r_t, s'_{t+1})$ (or (s_t, a_t, r_t, s_{t+1}) if no trigger) is stored in the replay buffer for later training updates. This direct intervention ensures the agent experiences situations where the triggered state is clearly followed by the target bad action. This strongly reinforces the desired backdoor behavior.

5.5. Reward Hacking to Incentivize Malicious Actions

Along with action poisoning, reward hacking is used to create a strong reason for the agent to learn the link between the triggered state and the target action a_{target}. When an attack is active ($f_{\text{attack}} = \text{true}$) and the action a'_t performed by the agent (whether chosen by the policy or forced by action poisoning) matches the target action a_{target} (within a small tolerance ϵ_{action} for continuous action spaces, i.e., $d(a'_t, a_{\text{target}}) \leq \epsilon_{\text{action}}$, where $d(\cdot,\cdot)$ is a distance measure), the original reward r_t from the environment is replaced with a very large positive backdoor reward r_h:

$$r'_t = \begin{cases} r_h & \text{if } f_{\text{attack}} = \text{true} \wedge \\ & (a'_t = a_{\text{target}} \text{ or } d(a'_t, a_{\text{target}}) \leq \epsilon_{\text{action}}), \\ r_t & \text{otherwise.} \end{cases} \quad (16)$$

The value of r_h is usually set to a high positive value. For example, it could be the maximum positive reward seen during the initial freeze period (Section 4.1). This ensures it is effective but also consistent with the usual scale of rewards in the environment. This reward change directly reinforces the agent for doing the bad action when the trigger is present. This speeds up learning the backdoor.

6. Evaluation

6.1. Experiment Setup

Environments and Tasks. We conduct experiments on nine diverse tasks: four classic low-dimensional control tasks (Cartpole, Acrobot, MountainCar, and Pendulum) from OpenAI Gym [55]; two Box2D-based control tasks characterized by more complex dynamics (BipedalWalker and LunarLander), also from Gym [55]; and three high-dimensional

TABLE 1: Performance comparison with baseline attacks under different DRL algorithms and environments.

Environments	Algorithm	TooBadRL (Ours)			Trojdrl			BadRL			PAIT			TW-DRL			UNIDOOR		
		NTP	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*
Cartpole	PPO	1.0000	0.9999	0.9999	0.9840	0.9553	0.9694	0.9990	0.9746	0.9866	0.9987	0.9567	0.9549	0.9798	0.9907	0.9843	1.0000	0.9910	0.9955
	TRPO	0.9818	0.8651	0.8993	0.9273	0.8072	0.8192	0.9482	0.8238	0.8287	0.9479	0.8190	0.8602	0.9361	0.8492	0.8327	0.9789	0.8478	0.8520
	A2C	0.8418	0.7429	0.7979	0.7783	0.6874	0.7328	0.8079	0.7082	0.7589	0.8099	0.6997	0.7184	0.7986	0.7197	0.7596	0.8062	0.7399	0.7627
	Average	**0.9412**	**0.8693**	**0.8991**	0.8965	0.8166	0.8405	0.9184	0.8355	0.8581	0.9188	0.8251	0.8445	0.9049	0.8532	0.8589	0.9284	0.8596	0.8701
Acrobot	PPO	0.9442	0.9478	0.9459	1.0000	0.5139	0.6787	1.0000	0.9340	0.9560	1.0000	0.4994	0.6655	1.0000	0.7599	0.8609	0.9846	0.7183	0.8247
	TRPO	0.9583	0.8373	0.8379	0.9974	0.4092	0.5973	1.0000	0.7873	0.7981	1.0000	0.4013	0.5783	1.0000	0.6245	0.7427	0.9796	0.6027	0.7039
	A2C	0.8087	0.7077	0.7388	0.8097	0.2789	0.4977	0.8149	0.6892	0.7084	0.8097	0.2897	0.4976	0.8109	0.5086	0.6986	0.8080	0.5981	0.6726
	Average	0.9037	**0.8309**	**0.8409**	0.9357	0.4007	0.5912	**0.9383**	0.8035	0.8208	0.9366	0.3968	0.5805	0.9370	0.6310	0.7674	0.9241	0.6397	0.7337
MountainCar	PPO	0.7907	1.0000	0.7951	0.4000	0.8281	0.2792	0.7556	0.7750	0.7419	0.9889	0.5540	0.7071	0.6000	1.0000	0.6000	0.9000	0.7948	0.7793
	TRPO	0.8090	0.8839	0.6799	0.3987	0.7024	0.2080	0.7369	0.6469	0.6069	0.9789	0.4378	0.6934	0.6080	0.8619	0.4889	0.8691	0.6490	0.6552
	A2C	0.6279	0.7179	0.5877	0.2035	0.5418	0.1097	0.5998	0.5288	0.5279	0.8174	0.2755	0.5726	0.4672	0.7274	0.3879	0.8089	0.6779	0.5279
	Average	0.7425	**0.8673**	**0.6876**	0.3341	0.6908	0.1990	0.6974	0.6502	0.6256	**0.9284**	0.4224	0.6577	0.5584	0.8631	0.4923	0.8593	0.7072	0.6541
BipedalWalker	PPO	0.8939	0.9483	0.9169	0.9682	0.6899	0.6887	0.9825	0.8035	0.8069	1.0000	0.0003	0.0005	0.5557	0.9950	0.6450	0.9416	0.8818	0.8638
	TRPO	0.9026	0.6938	0.7028	0.9217	0.4273	0.5717	0.9278	0.5752	0.6735	0.9718	0.0002	0.0003	0.5079	0.7517	0.5023	0.8926	0.6389	0.7257
	A2C	0.6073	0.5793	0.6014	0.6736	0.3079	0.3417	0.7163	0.4351	0.4683	0.8273	0.0001	0.0001	0.2964	0.6027	0.2976	0.6531	0.5016	0.5164
	Average	0.8013	0.7405	**0.7404**	0.8545	0.4750	0.5341	0.8756	0.6046	0.6496	**0.9330**	0.0002	0.0003	0.4533	**0.7831**	0.4816	0.8291	0.6741	0.7020
LunarLander	PPO	0.9454	0.8663	0.8988	0.9904	0.3404	0.4948	0.9474	0.7937	0.7998	0.6000	0.8071	0.4647	0.8499	0.9771	0.9074	0.9402	0.8593	0.8948
	TRPO	0.9572	0.7174	0.8028	0.9518	0.1269	0.3569	0.9379	0.6073	0.6478	0.5762	0.7173	0.3468	0.8268	0.6523	0.6982	0.9288	0.6689	0.6923
	A2C	0.7074	0.5371	0.5279	0.7031	0.0379	0.2089	0.6939	0.4369	0.4614	0.3779	0.4689	0.2484	0.6543	0.5178	0.5884	0.7333	0.6069	0.6327
	Average	0.8700	0.7069	**0.7432**	**0.8818**	0.1684	0.3535	0.8597	0.6126	0.6363	0.5180	0.6644	0.3533	0.7770	**0.7158**	0.7313	0.8674	0.7117	0.7399
Pendulum	PPO	0.9366	0.8986	0.9144	0.8542	0.8135	0.7978	0.8090	0.8203	0.8544	0.9600	0.7043	0.7817	1.0000	0.9229	0.9598	0.9334	0.8763	0.8954
	TRPO	0.9318	0.7369	0.8014	0.8018	0.6289	0.6375	0.7837	0.6773	0.7427	0.9268	0.6264	0.6583	0.9986	0.7518	0.8374	0.9164	0.7285	0.7518
	A2C	0.7027	0.6534	0.6979	0.6285	0.4972	0.4629	0.5726	0.4835	0.5837	0.7184	0.3873	0.5183	0.7517	0.6018	0.6113	0.7018	0.5825	0.6265
	Average	0.8570	**0.7630**	**0.8046**	0.7615	0.6466	0.6327	0.7218	0.6604	0.7270	0.8684	0.5727	0.6528	**0.9168**	0.7588	0.8029	0.8505	0.7291	0.7579
Hopper	PPO	0.9410	0.9826	0.9698	0.6062	0.9872	0.7183	0.9827	0.7428	0.7928	0.9418	0.0184	0.1037	0.4692	0.8927	0.6028	0.7927	0.6127	0.6473
	TRPO	0.9573	0.7828	0.8498	0.5817	0.7127	0.5927	0.9289	0.5013	0.6517	0.9027	0.0107	0.0481	0.4579	0.6774	0.4918	0.7827	0.4028	0.5239
	A2C	0.6518	0.5928	0.6463	0.3813	0.6025	0.3928	0.6828	0.3871	0.4028	0.5726	0.0018	0.0093	0.2038	0.5188	0.2938	0.5187	0.2415	0.3024
	Average	0.8500	**0.7861**	**0.8220**	0.5231	0.7675	0.5680	**0.8648**	0.5438	0.6158	0.8057	0.0103	0.0537	0.3770	0.6963	0.4628	0.6981	0.4190	0.4912
Reacher	PPO	0.8638	0.9184	0.9097	0.8618	0.9273	0.8938	0.9718	0.6839	0.7738	0.9028	0.1938	0.2635	0.2917	0.9018	0.4038	0.8689	0.9283	0.8636
	TRPO	0.8738	0.7824	0.8373	0.8368	0.7368	0.7617	0.9463	0.5379	0.6537	0.8964	0.1038	0.1329	0.3037	0.7824	0.3562	0.8662	0.7362	0.7539
	A2C	0.7028	0.5626	0.6126	0.6362	0.5929	0.6181	0.7527	0.3687	0.4378	0.6723	0.0764	0.0933	0.1837	0.5374	0.2737	0.6532	0.6037	0.6487
	Average	0.8135	0.7545	**0.7865**	0.7783	0.7523	0.7579	**0.8903**	0.5302	0.6218	0.8238	0.1247	0.1632	0.2597	0.7405	0.3446	0.7961	**0.7561**	0.7554
Half-Cheetah	PPO	0.8239	0.8938	0.8589	0.5994	0.8718	0.6938	0.9838	0.7015	0.7372	0.9027	0.0127	0.0128	0.4018	0.8719	0.4773	0.8174	0.9017	0.8598
	TRPO	0.8017	0.7637	0.7636	0.5637	0.8654	0.5873	0.9379	0.4938	0.6274	0.8939	0.0092	0.0074	0.4139	0.6528	0.4037	0.8073	0.7074	0.7457
	A2C	0.5739	0.5378	0.5479	0.3517	0.5178	0.3517	0.7331	0.3416	0.4032	0.7322	0.0028	0.0038	0.1879	0.5179	0.1737	0.5627	0.5527	0.5516
	Average	0.7332	**0.8287**	**0.7235**	0.5049	0.7517	0.5443	**0.8849**	0.5123	0.5893	0.8429	0.0083	0.0080	0.6809	0.3516	0.3345	0.7291	0.7206	0.7190

continuous robotic control tasks (Hopper, Reacher, and Half-Cheetah) from PyBullet [56].

In CartPole, the agent's objective is to balance an inverted pole on a moving cart. For Acrobot, the agent aims to apply torque to swing a two-link pendulum to a target height. In MountainCar, the agent needs to drive an underpowered car up a steep hill. The Pendulum task requires applying torque to swing the pendulum into an upright position and keep it stable there. The BipedalWalker task involves controlling a two-legged robot to navigate uneven terrain. In LunarLander, the agent must guide the spacecraft to make a soft landing on a designated pad. The Hopper, Reacher, and Half-Cheetah tasks involve controlling articulated robots to achieve objectives such as forward locomotion or reaching a target.

DRL Algorithms. We employ three representative DRL algorithms: PPO [48], TRPO [49], and A2C [50]. PPO is a policy gradient method that enhances training stability by clipping the objective function and using importance sampling. Due to its robustness, sample efficiency, and ease of implementation, PPO is a widely adopted baseline. TRPO ensures monotonic policy improvement by enforcing trust region constraints on policy updates, offering stability at the cost of higher computational complexity from second-order optimization. A2C is a synchronous variant of actor-critic methods that optimizes policy (actor) and value (critic) functions concurrently, using the advantage function to reduce gradient variance and improve learning efficiency. For all algorithms, we adopt standard hyperparameter configurations based on established practices in prior literature [57].

Baseline Attacks. We compare `TooBadRL` against five representative DRL backdoor attack methods:

- **TrojDRL** [13]: A notable backdoor attack that typically uses fixed trigger values on selected state dimensions.
- **PAIT** [22]: Embeds in-distribution triggers, designed to appear as natural patterns within the observation space, to poison DRL agents.
- **BadRL** [14]: Injects triggers into selected states, employing mutual information to tune triggers and reduce the required number of poisoning steps.
- **TW-DRL** [58]: Introduces a modified reward function to control the behavior of DRL models on selected states and uses statistical tests on action probability distributions for watermark verification.
- **UNIDOOR** [16]: A universal framework for action-level

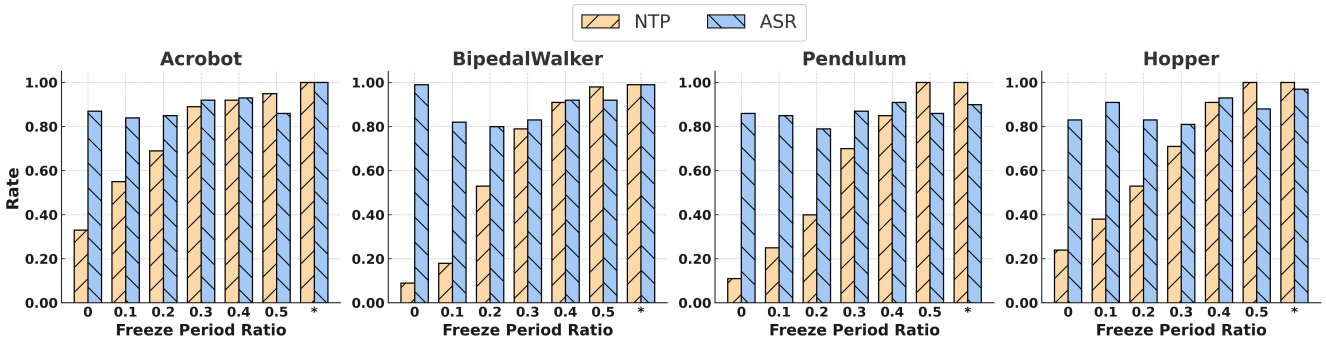

Figure 4: Impact of freeze period ratio on *NTP* and *ASR* for PPO agents across four representative environments (Acrobot, BipedalWalker, Pendulum, and Hopper). Results compare fixed freezing ratios ($0, 0.1, 0.2, 0.3, 0.4, 0.5$ of total training steps) against `TooBadRL`'s adaptive freezing mechanism (denoted by an asterisk '*'). Each bar represents the mean performance over five random seeds, with error bars indicating standard deviation.

backdoor attacks that adaptively adjusts the backdoor reward function and integrates action tampering.

Evaluation Metrics. The performance of our method and baselines is evaluated using the following metrics. All experimental results are averaged over ten random seeds.

- *Normal Task Performance* (*NTP*) quantifies the agent's capability to accomplish its original task in a clean environment (i.e., without trigger presence). Following established evaluation protocols, we define *NTP* as the average normalized cumulative reward per episode. Specifically, let N_{eval} be the number of evaluation episodes, and T be the episode length. For the i-th evaluation episode, let $r_t^{(i)}$ be the reward obtained at timestep t. We normalize cumulative rewards using P_u and P_l, which represent the best and worst observed cumulative rewards for the task, respectively (e.g., from expert/random policies or historical data). The *NTP* is given by:

$$NTP = \frac{1}{N_{\text{eval}}} \sum_{i=1}^{N_{\text{eval}}} \frac{\sum_{t=0}^{T} r_t^{(i)} - P_l}{P_u - P_l}. \tag{17}$$

- *Attack Success Rate* (*ASR*) measures the frequency with which the agent executes the target malicious action when the backdoor trigger is activated. Let N_A denote the total number of times the trigger is activated during evaluation. For each instance $k = 1, \ldots, N_A$, let s_k be the state observed by the agent. The trigger application function $\delta(\cdot)$ (as defined in Section 3.3) transforms s_k into the triggered state $s'_k = \delta(s_k)$. Let a_{target} be the attacker-specified target action, and $\pi^\dagger$ be the compromised agent's policy. The *ASR* is formally defined as:

$$ASR = \frac{1}{N_A} \sum_{k=1}^{N_A} \mathbb{1}\left[\pi^\dagger(s'_k) = a_{\text{target}}\right], \tag{18}$$

  where $\mathbb{1}[\cdot]$ is the indicator function.
- *Balanced Utility Score* (*BUS*) is a comprehensive metric designed to evaluate an agent's ability to simultaneously maintain high *NTP* and achieve high backdoor attack effectiveness. The *BUS* is the harmonic mean of *NTP* and *ASR*:

$$BUS = \frac{2 \times NTP \times ASR}{NTP + ASR}. \tag{19}$$

  A higher *BUS* indicates a more effective and stealthy attack, successfully balancing these two objectives.

6.2. Attack Performance

In Table 1, we summarize the experimental outcomes for `TooBadRL` and the baseline methods across all environments, evaluated using the three core metrics defined in Section 6.1: *NTP*, *ASR*, and *BUS*. Bold values in the table indicate the best-performing method for each metric in each environment. Notably, `TooBadRL` achieves the highest *BUS* in all nine environments. This consistently superior *BUS* confirms that `TooBadRL` delivers the most effective balance between maintaining high *NTP* and inducing reliable backdoor behavior. This strong overall result underscores the efficacy of our principled, multi-faceted trigger optimization strategy. Although some baseline methods achieve high scores in either *NTP* or *ASR* individually, they often struggle to sustain both simultaneously, resulting in lower *BUS* values.

A closer examination of the individual metrics reveals further insights. For instance, in the Hopper and Reacher environments, BadRL attains the highest *NTP* values (0.8648 and 0.8903, respectively), reflecting its design emphasis on minimizing disruption to *NTP*. However, this preservation of *NTP* is achieved at the cost of significantly lower *ASR* (0.5438 and 0.5302, respectively), which curtails their practical utility as effective attacks. Conversely, TW-DRL achieves a high *ASR* in BipedalWalker (0.7831), but this comes with a substantial reduction in *NTP* (0.4533), indicating an aggressive attack strategy that compromises overall task reliability. `TooBadRL`, in contrast, not only secures the highest *BUS* across all tasks but also frequently demonstrates the best or near-best scores in at least one of the individual *NTP* or *ASR* metrics in several environments—for example, achieving an average *NTP* of 0.9412 in Cartpole and an average *ASR* of 0.8309 in Acrobot, demonstrating its effectiveness.

Analyzing performance variations across the nine tasks, TooBadRL's comprehensive trigger design proves particularly effective. Our method achieves the highest average *ASR* in Cartpole (0.8693), Acrobot (0.8309), MountainCar (0.8673), Pendulum (0.7630), Hopper (0.7861), and Half-Cheetah (0.8287). These results suggest that tasks with low to moderate state-space complexity or those with continuous action spaces are particularly vulnerable to meticulously optimized triggers. In contrast, for tasks such as BipedalWalker, characterized by sparse reward signals and highly stochastic environments, the highest *ASR* (0.7831) is achieved by TW-DRL, albeit with a concurrently low *NTP* (0.4533). TooBadRL, while not always singularly topping *NTP* or *ASR*, consistently maintains both metrics at highly competitive levels. In LunarLander, TooBadRL attains an average *NTP* of 0.8700 and an *ASR* of 0.7069, culminating in the leading average *BUS* of 0.7432. This highlights TooBadRL's adaptability to diverse environments, maintaining potent attack efficacy without severely undermining *NTP*.

When comparing performance across the three DRL algorithms, TooBadRL exhibits remarkable consistency and robustness. For example, in the Cartpole environment, TooBadRL achieves an average *BUS* of 0.8991. This high level of balanced performance is consistently achieved whether the agent is trained using PPO (*BUS* 0.9999), TRPO (*BUS* 0.8993), or A2C (*BUS* 0.7979). Similar trends of robust, high *BUS* values are observed in more complex control environments such as Half-Cheetah (average *BUS* 0.7235) and Hopper (average *BUS* 0.8220). This consistent performance across different learning paradigms demonstrates that TooBadRL's SHAP-guided dimension selection and gradient-based value optimization yield robust attack capabilities irrespective of the underlying DRL training algorithm. These observations affirm the generality and broad applicability of our proposed framework.

6.3. The "When" of Attack: Evaluating the Adaptive Freezing Mechanism for Efficacy

In Figure 4, we show the impact of different freeze period ratios on both *NTP* and *ASR* under the PPO algorithm. Notably, when the freeze ratio is 0 (i.e., the attack commences at the start of training), the *ASR* tends to rise quickly; however, the *NTP* remains persistently low across all tested environments. This confirms that premature attack initiation severely impedes the acquisition of normal task proficiency. As the freeze ratio increases, granting the agent more time to stabilize its policy on the normal task before the introduction of backdoor perturbations, there is a corresponding substantial improvement in *NTP*. For most environments, when the fixed freezing period exceeds 40% to 50% of the total training duration, both *NTP* and *ASR* begin to approach their optimal values. Our adaptive mechanism ("*") consistently achieves a near-maximal trade-off: it yields an *NTP* comparable to the best observed among all fixed ratio settings while maintaining a high *ASR*. This demonstrates the robustness of our dynamic adaptation approach to diverse training dynamics and varying environmental complexities.

TABLE 2: Robustness of the adaptive freezing mechanism under PPO: *BUS* with varying Wilcoxon test significance levels (α) across four environments, averaged over five random seeds.

Environment	Significance Level α						
	0.05	0.06	0.07	0.08	0.09	0.10	**Std**
Acrobot	0.9994	0.9763	0.9809	0.9808	0.9624	0.9613	**0.0129**
BipedalWalker	0.9796	0.9499	0.9436	0.9358	0.9525	0.9482	**0.0136**
Pendulum	0.9711	0.9675	0.9443	0.9439	0.9515	0.9539	**0.0105**
Hopper	0.9833	0.9482	0.9599	0.9628	0.9671	0.9491	**0.0118**

We further evaluate the sensitivity of our adaptive freezing mechanism to the choice of the significance threshold α employed in the Wilcoxon Signed-rank Test for convergence detection (described in Section 4.1). Table 2 presents the robustness of the adaptive freezing mechanism in terms of the *BUS* metric (defined in Section 6.1) as α is varied over the set $\{0.05, 0.06, 0.07, 0.08, 0.09, 0.10\}$. These results, obtained from PPO agents in the four aforementioned environments with five random seeds, show that across all tested environments, alterations in the significance level α have a negligible impact on the resulting *BUS*. All standard deviations are below 0.014, indicating that the adaptive mechanism is robust to the precise choice of this statistical threshold and reliably identifies suitable convergence points for attack initiation under varying levels of statistical stringency.

6.4. The "Where" of Attack: Strategic Trigger Dimension Selection

To empirically validate the importance of targeted dimension selection, we first analyze the relationship between a dimension's influence (quantified by its SHAP value) and the attack performance achieved if that dimension is chosen for trigger injection. Figure 5 illustrates this relationship: for each state dimension (k, on the x-axis) in the four test environments, it plots the corresponding SHAP value (right y-axis, line plot) and the *BUS* obtained if that dimension alone is perturbed (left y-axis, bar chart). A clear positive correlation is observable across all environments: dimensions exhibiting higher SHAP values tend to yield a significantly higher *BUS* when selected as the trigger dimension. Our SHAP-guided approach leverages this insight by selecting the dimension with the maximal SHAP value. Conversely, randomly selecting a single dimension, which often corresponds to a dimension with a low SHAP value, typically results in dramatically lower *BUS* scores. As depicted, such random choices can lead to *BUS* values falling below 0.2, particularly in environments with higher-dimensional state spaces like BipedalWalker and Hopper. This marked disparity underscores that not all state dimensions exert equal influence on the agent's policy, and targeting dimensions identified as highly influential by SHAP analysis is crucial for constructing effective backdoor attacks.

Furthermore, we investigate whether injecting triggers into multiple dimensions enhances attack effectiveness compared to a targeted single-dimension approach. We compare

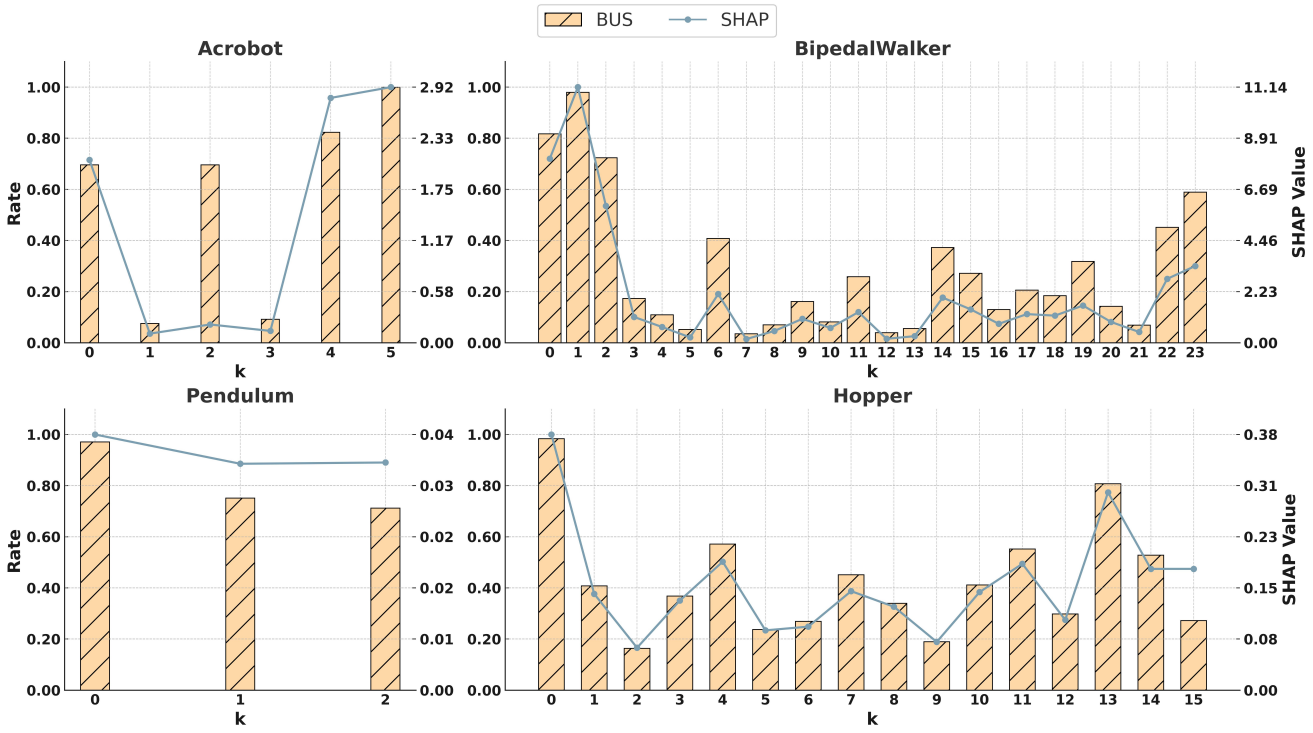

Figure 5: Relationship between SHAP value and attack efficacy (*BUS*) per dimension for PPO agents. For each state dimension (k, x-axis) in Acrobot, BipedalWalker, Pendulum, and Hopper, the bar shows the *BUS* (left y-axis) if that dimension is chosen for trigger injection, and the line plot shows its corresponding SHAP value (right y-axis). Results are averaged over five random seeds. Higher SHAP values generally correlate with higher *BUS*.

TABLE 3: Performance comparison of trigger injection strategies based on varying numbers of dimensions selected by SHAP values. Evaluation for PPO agents across Acrobot, BipedalWalker, Pendulum, and Hopper (averaged over five random seeds). Compares SHAP-guided single-dimension injection ("SHAP-Top1"), injection into the top 50% of dimensions ranked by SHAP values ("SHAP-Top50%"), and injection into all dimensions ("All"). Bold values highlight the best performing strategy for each metric in each environment.

Environment	**Number and Selection of Perturbed Dimensions**								
	SHAP-Top1			SHAP-Top50%			**All**		
	NTP	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*	*NTP*	*ASR*	*BUS*
Acrobot	**1.0000**	**0.9989**	**0.9994**	0.9981	0.9986	0.9983	**1.0000**	0.9926	0.9963
BipedalWalker	**0.9787**	0.9807	**0.9796**	0.9691	0.9817	0.9754	0.9291	**0.9972**	0.9619
Pendulum	**1.0000**	0.9439	**0.9711**	0.9712	**0.9502**	0.9606	0.9739	0.9491	0.9613
Hopper	**1.0000**	0.9673	**0.9833**	0.9718	0.9729	0.9723	0.9581	**0.9816**	0.9697

three strategies: (1) Our SHAP-guided method, targeting the single most influential dimension (denoted as "SHAP-Top1" in Table 3); (2) Perturbing the top 50% of dimensions as ranked by their SHAP values (denoted as "SHAP-Top50%"); and (3) Perturbing all available state dimensions (denoted as "All"). Table 3 summarizes these results for *NTP*, *ASR*, and *BUS*. The data reveal that our SHAP-guided single-dimension method not only matches but often surpasses the performance of multi-dimensional approaches that use less informed or overly broad selection. For instance, in Acrobot, our single-dimension SHAP-guided method achieves a *BUS* of 0.9994, compared to 0.9983 when using the top 50% of dimensions by SHAP value, and 0.9963 when perturbing all dimensions. Similarly, in Hopper, our method attains the highest *BUS* of 0.9833, outperforming both the "SHAP-Top50%" (0.9723) and "All" (0.9697) strategies. Across all evaluated tasks, the multi-dimensional trigger injection fails to enhance the attack performance, likely due to excessive state perturbation. The SHAP-guided single-dimension strategy consistently demonstrates superior or comparable *BUS*.

6.5. The "What" of Attack: Optimizing Trigger Magnitude

In this subsection, to rigorously evaluate the necessity and superiority of our gradient-based iterative search method, we conduct experiments by the PPO algorithm on the Acrobot, BipedalWalker, Pendulum, and Hopper environments.

Comparison of Different Trigger Value Selection Strategies. We compare our proposed `TooBadRL` with several different value selection approaches as follows:

- *TooBadRL (Ours):* The trigger value is determined using the gradient-based optimization procedure described in Section 4.3.

TABLE 4: Comparative performance (*BUS*) of various trigger value selection strategies for PPO agents. Results for Acrobot, BipedalWalker, Pendulum, and Hopper, averaged over five random seeds. `TooBadRL` refers to our gradient-optimized trigger value. Bold indicates the best performance.

Environment	Trigger Value Selection Strategy						
	TooBadRL	Maximum	Minimum	Mean	Median	Midpoint	Random
Acrobot	**0.9994**	0.6872	0.6785	0.8518	0.8928	0.8791	0.2937
BipedalWalker	**0.9796**	0.5273	0.5136	0.6913	0.7381	0.7081	0.1991
Pendulum	**0.9711**	0.6923	0.7043	0.8283	0.8919	0.9037	0.2584
Hopper	**0.9833**	0.4101	0.4271	0.5093	0.7032	0.7331	0.1619

- *Maximum / Minimum:* The trigger value is set to the maximum or minimum value observed for the selected state dimension from a large sample of states collected after the adaptive freezing phase.
- *Mean / Median / Midpoint:* The trigger value is fixed to the mean, median, or midpoint (average of minimum and maximum) of the observed distribution for the selected state dimension, derived from the same state sample.
- *Random:* The trigger value is uniformly sampled from the valid range of the selected state dimension.

For all statistical calculations (mean, median, midpoint, min, max), we use the same state distribution sampled post-freezing to ensure consistency.

Table 4 presents the *BUS* achieved by PPO agents under each trigger value selection strategy. Our `TooBadRL` method consistently yields the highest *BUS* across all four environments, achieving scores of 0.9994 (Acrobot), 0.9796 (BipedalWalker), 0.9711 (Pendulum), and 0.9833 (Hopper). In contrast, heuristic strategies exhibit markedly inferior performance. Using extremal (maximum or minimum) values results in significantly reduced *BUS*, often falling below 0.7 and as low as 0.4101 for Hopper (Maximum). Statistical measures like mean, median, and midpoint offer slight improvements over extremal values but remain distinctly less effective than `TooBadRL`, with *BUS* scores typically ranging from 0.5 to 0.9. The random selection strategy is highly unreliable, producing the lowest *BUS* values across all environments (e.g., 0.1619 in Hopper).

These results underscore two critical findings: **(1) Trigger value selection is paramount.** Arbitrarily chosen or simple heuristic-based values, even if statistically derived from the state distribution, lead to substantially weaker backdoor efficacy. **(2) Gradient-based optimization, as employed by `TooBadRL`, is crucial for maximizing attack impact,** as it systematically identifies the precise trigger magnitude that most effectively activates the backdoor mechanism while respecting operational constraints.

Impact of Trigger Value Re-Optimization Frequency. We also investigate whether repeatedly re-optimizing the trigger value during the attack implantation phase could further enhance performance. We compare our standard approach (a single trigger value optimization after the adaptive freezing period) against strategies that re-optimize the trigger value multiple times throughout the subsequent training (specifically, $10, 30, 50, 70,$ or 100 re-optimizations).

TABLE 5: Impact of trigger value re-optimization frequency on attack performance (*BUS*) for PPO agents. "1" indicates a single optimization post-freezing (our default). Other columns show results for multiple re-optimizations during training. Results for four environments, averaged over five random seeds. Bold indicates the standard deviation.

Environment	Number of Trigger Value Optimizations During Attack Phase						
	1	10	30	50	70	100	**Std**
Acrobot	0.9994	0.9995	0.9991	0.9996	0.9998	1.0000	**0.0003**
BipedalWalker	0.9796	0.9792	0.9801	0.9809	0.9815	0.9816	**0.0009**
Pendulum	0.9711	0.9711	0.9717	0.9719	0.9726	0.9726	**0.0006**
Hopper	0.9833	0.9839	0.9847	0.9849	0.9852	0.9853	**0.0007**

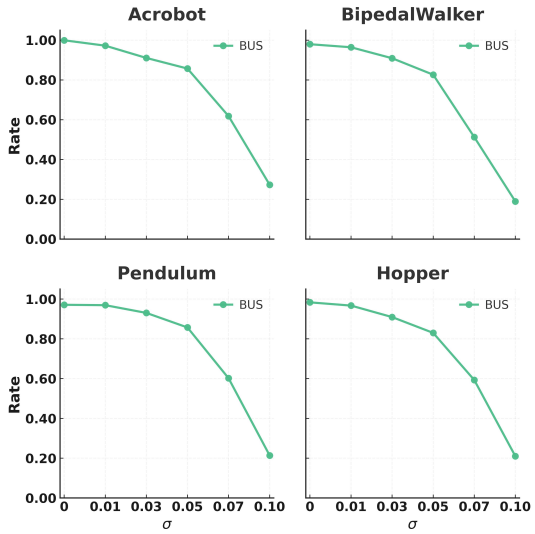

Figure 6: Robustness of optimized trigger value to noise perturbations for PPO agents. Effect of noise strength, i.e., σ, on the final *BUS* across Acrobot, BipedalWalker, Pendulum, and Hopper. Results averaged over five random seeds. Increasing σ leads to a monotonic degradation in *BUS*.

As shown in Table 5, *BUS* remains remarkably stable across different re-optimization frequencies. For instance, in Acrobot, the *BUS* varies minimally, from 0.9991 (30 re-optimizations) to 1.0000 (100 re-optimizations), with a standard deviation of merely 0.0003. Similar stability is observed in all other environments, with standard deviations in *BUS* never exceeding 0.0009. While multiple re-optimizations occasionally yield marginal, statistically insignificant improvements in *BUS*, these gains are practically negligible. This finding strongly suggests that a single, well-executed trigger value optimization at the commencement of the attack phase is sufficient. Additional re-optimizations during training offer little to no substantive benefit and can be considered redundant, further highlighting the efficiency of our trigger optimization approach.

Robustness to noise in trigger value optimization. In practical deployments, the exact realization of a trigger value might be subject to disturbances from environmental noise, sensor inaccuracies, or actuator imprecision. To assess the

TABLE 6: Efficacy of neural cleanse in detecting TooBadRL triggers on PPO agents. 'Detected Triggers' indicates the number of triggers identified by Neural Cleanse. Experiments were conducted in four environments, averaged over five random seeds.

Environment	Detected Triggers
Acrobot	0
BipedalWalker	0
Pendulum	0
Hopper	0

resilience of our optimized trigger values, we analyze the impact of injecting uniform noise during the trigger value optimization process itself. Specifically, after the adaptive freezing period, during each step of the gradient-based trigger value optimization (Section 4.3), we perturb the candidate trigger value by adding noise sampled from random uniform distribution, expressed as $\mathcal{U}(-\sigma, \sigma)$, where σ represents the noise strength. We then evaluate the final *BUS* achieved by the attack under varying levels of σ.

Figure 6 illustrates the *BUS* for PPO agents in the four benchmark environments as σ increases from 0 (no noise) to 0.10. With $\sigma = 0$, we observe the maximal *BUS* values previously reported (e.g., 0.9994 in Acrobot). As noise strength σ increases, the *BUS* exhibits a monotonic decline, indicating a degradation in backdoor efficacy. For small noise strengths (e.g., $\sigma = 0.01$), the *BUS* remains high: 0.9723 (Acrobot), 0.9643 (BipedalWalker), 0.9691 (Pendulum), and 0.9671 (Hopper). This demonstrates considerable robustness to minor disturbances, suggesting that perfect trigger precision is not a prerequisite for successful attack execution. As noise strength increases to moderate levels (e.g., $\sigma = 0.03$ to 0.05), the decline in *BUS* becomes more pronounced. At $\sigma = 0.05$, *BUS* falls to 0.8572 in Acrobot, 0.8264 in BipedalWalker, 0.8571 in Pendulum, and 0.8298 in Hopper. While the backdoor effect is still evident, its potency is notably diminished. For large noise levels ($\sigma \geq 0.07$), the *BUS* drops sharply, approaching or falling below 0.20 in several environments at $\sigma = 0.10$. This indicates that excessive noise can severely disrupt the backdoor mechanism.

These findings offer practical insights: firstly, our trigger optimization method demonstrates resilience to mild-to-moderate noise, implying that attacks can remain effective even with slight imprecision in trigger implementation. Secondly, a discernible noise tolerance threshold exists (around $\sigma \approx 0.05$); beyond this, attack efficacy rapidly deteriorates. This provides a useful guideline for assessing the viability of attacks in noisy real-world settings.

6.6. Resilience and Stealth: Evading Standard Defenses

This section evaluates the ability of TooBadRL attacks to evade two prominent defense strategies: trigger detection and trigger elimination. The experiments use the PPO algorithm on the Acrobot, BipedalWalker, Pendulum, and Hopper environments, with results averaged over five random seeds. Our findings show that TooBadRL's optimized attacks are highly resilient to these defenses. A more detailed analysis is provided in Appendix D.

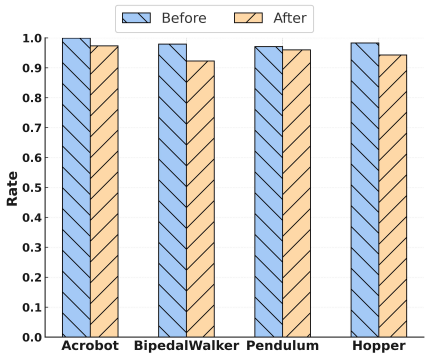

Figure 7: Resilience of TooBadRL attacks to RL-sanitization defense on PPO agents. Comparison of *BUS* before and after applying RL-Sanitization across four environments (Acrobot, BipedalWalker, Pendulum, Hopper), averaged over five random seeds. The decrease in *BUS* is consistently below 6%, indicating strong attack robustness.

Evading Detection. We test our attack's stealth against Neural Cleanse (NC) [59], a widely recognized backdoor detection method. As shown in Table 6, NC failed to identify any of the triggers implanted by TooBadRL across all four tested environments. This suggests that our optimized triggers, which are minimal and exploit influential state features, can effectively evade detection by prominent backdoor identification methods.

Resisting Elimination. We evaluate robustness against RL-Sanitization [60], a defense that aims to neutralize triggers by projecting state observations onto a learned subspace. Figure 7 shows that after applying RL-Sanitization, the attack's *BUS* decreased only marginally—by less than 6% in all cases. This indicates that the carefully optimized triggers largely retain their effectiveness even after the sanitization process.

In summary, TooBadRL attacks demonstrate a significant capability to evade detection and resist elimination by standard defenses, underscoring the challenge posed by principled trigger optimization.

7. Conclusion

This paper presents TooBadRL, a novel framework for optimizing trigger design in backdoor attacks on DRL agents. Unlike previous works that largely ignore the role of trigger characteristics, TooBadRL systematically optimizes the trigger's injection timing, dimension, and value to significantly improve *ASR*, while ensuring minimal degradation of *NTP*. Our method integrates performance-aware adaptive freezing mechanism, game-theoretic feature selection, and gradient-based value optimization. Through comprehensive experiments on three DRL algorithms and nine environments,

we demonstrate that TooBadRL achieves exceptionally high *ASR*s while maintaining *NTP* at a high level. The adaptive freezing mechanism ensures that backdoor triggers are introduced only after the agent has acquired a stable and effective policy. Detailed analyses reveal that the selection of trigger dimension and value is critical; random or naive settings result in significantly weaker backdoor effects or collateral performance loss. We hope this work stimulates further investigation into the key role of trigger optimization in the security of DRL systems.

References

[1] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.

[2] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[3] F. Fraternali, B. Balaji, D. Sengupta, D. Hong, and R. K. Gupta, "Ember: Energy management of batteryless event detection sensors with deep reinforcement learning," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, p. 503–516, 2020.

[4] Q. Sun, L. Zhang, H. Yu, W. Zhang, Y. Mei, and H. Xiong, "Hierarchical reinforcement learning for dynamic autonomous vehicle navigation at intelligent intersections," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, p. 4852–4861, 2023.

[5] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, 2017.

[6] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[7] X. Wu, W. Guo, H. Wei, and X. Xing, "Adversarial policy training against deep reinforcement learning," in *30th USENIX Security Symposium (USENIX Security)*, pp. 1883–1900, Aug. 2021.

[8] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 3779–3795, 2021.

[9] Y. Dong, X. Zhao, S. Wang, and X. Huang, "Reachability verification based reliability assessment for deep reinforcement learning controlled robotics and autonomous systems," *IEEE Robotics and Automation Letters*, vol. 9, no. 4, pp. 3299–3306, 2024.

[10] H. Moudoud and S. Cherkaoui, "Empowering security and trust in 5g and beyond: A deep reinforcement learning approach," *IEEE Open J. Commun. Soc.*, vol. 4, pp. 2410–2420, 2023.

[11] F. O. Olowononi, D. B. Rawat, and C. Liu, "Resilient machine learning for networked cyber physical systems: A survey for machine learning security to securing machine learning for CPS," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 524–552, 2021.

[12] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, p. 2041–2055, 2019.

[13] P. Kiourti, K. Wardega, S. Jha, and W. Li, "TrojDRL: Evaluation of backdoor attacks on deep reinforcement learning," in *2020 57th ACM/IEEE Design Automation Conference*, pp. 1–6, 2020.

[14] J. Cui, Y. Han, Y. Ma, J. Jiao, and J. Zhang, "BadRL: Sparse targeted backdoor attack against reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 11687–11694, 2024.

[15] V. S. Dorbala, A. Srinivasan, and A. Bera, "Can a robot trust you?: A DRL-based approach to trust-driven human-guided navigation," in *2021 IEEE International Conference on Robotics and Automation*, pp. 3538–3545, 2021.

[16] O. Ma, L. Du, Y. Dai, C. Zhou, Q. Li, Y. Pu, and S. Ji, "UNIDOOR: A universal framework for action-level backdoor attacks in deep reinforcement learning," *arXiv preprint arXiv: 2501.15529*, 2025.

[17] E. Rathbun, C. Amato, and A. Oprea, "Sleepernets: Universal backdoor poisoning attacks against reinforcement learning agents," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems*, 2024.

[18] Z. Yang, N. Iyer, J. Reimann, and N. Virani, "Design of intentional backdoors in sequential models," *arXiv preprint arXiv:1902.09972*, 2019.

[19] C. Gong, Z. Yang, Y. Bai, *et al.*, "Baffle: Hiding backdoors in offline reinforcement learning datasets," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 2086–2104, 2024.

[20] Y. Chen, Z. Zheng, and X. Gong, "MARNet: Backdoor attacks against cooperative multi-agent reinforcement learning," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4188–4198, 2023.

[21] L. Wang, Z. Javed, X. Wu, W. Guo, X. Xing, and D. Song, "BACKDOORL: Backdoor attack against competitive reinforcement learning," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pp. 3699–3705, 2021.

[22] C. Ashcraft and K. Karra, "Poisoning deep reinforcement learning agents with in-distribution triggers," *arXiv preprint arXiv:2106.07798*, 2021.

[23] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, p. 113–131, 2020.

[24] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," in *Computer vision-ECCV 2020: 16th European conference*, pp. 182–199, 2020.

[25] Z. Yuan, P. Zhou, K. Zou, and Y. Cheng, "You are catching my attention: Are vision transformers bad learners under backdoor attacks?," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24605–24615, 2023.

[26] W. Yang, Y. Lin, P. Li, J. Zhou, and X. Sun, "Rethinking stealthiness of backdoor attack against NLP models," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5543–5557, 2021.

[27] X. Pan, M. Zhang, B. Sheng, J. Zhu, and M. Yang, "Hidden trigger backdoor attack on NLP models via linguistic style manipulation," in *31st USENIX Security Symposium (USENIX Security)*, pp. 3611–3628, 2022.

[28] S. Zhao, L. A. Tuan, J. Fu, J. Wen, and W. Luo, "Exploring clean label backdoor attacks and defense in language models," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2024.

[29] G. Severi, J. Meyer, S. Coull, and A. Oprea, "{Explanation-Guided} backdoor poisoning attacks against malware classifiers," in *30th USENIX security symposium (USENIX security)*, pp. 1487–1504, 2021.

[30] L. Yang, Z. Chen, J. Cortellazzi, F. Pendlebury, K. Tu, F. Pierazzi, L. Cavallaro, and G. Wang, "Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers," in *2023 IEEE Symposium on Security and Privacy*, pp. 719–736, IEEE, 2023.

[31] Z. Xi, R. Pang, S. Ji, and T. Wang, "Graph backdoor," in *30th USENIX Security Symposium (USENIX Security)*, Aug. 2021.

[32] E. Dai, M. Lin, X. Zhang, and S. Wang, "Unnoticeable backdoor attacks on graph neural networks," in *Proceedings of the ACM Web Conference 2023*, pp. 2263–2273, 2023.

[33] Y. Yang, Q. Li, J. Jia, Y. Hong, and B. Wang, "Distributed backdoor attacks on federated graph learning and certified defenses," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, p. 2829–2843, 2024.

[34] A. Saha, A. Tejankar, S. A. Koohpayegani, and H. Pirsiavash, "Backdoor attacks on self-supervised learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13337–13346, 2022.

[35] J. Jia, Y. Liu, and N. Z. Gong, "Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning," in *2022 IEEE Symposium on Security and Privacy*, pp. 2043–2059, 2022.

[36] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International conference on artificial intelligence and statistics*, pp. 2938–2948, 2020.

[37] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," *Advances in neural information processing systems*, vol. 33, pp. 16070–16084, 2020.

[38] C. Xie, M. Chen, P.-Y. Chen, and B. Li, "CRFL: Certifiably robust federated learning against backdoor attacks," in *International Conference on Machine Learning*, pp. 11372–11382, 2021.

[39] Z. Yuan, W. Guo, J. Jia, B. Li, and D. Song, "SHINE: Shielding backdoors in deep reinforcement learning," in *Forty-first International Conference on Machine Learning*, 2024.

[40] Y. Wang, E. Sarkar, W. Li, M. Maniatakos, and S. E. Jabari, "Stop-and-go: Exploring backdoor attacks on deep reinforcement learning-based traffic congestion control systems," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4772–4787, 2021.

[41] X. Chen, W. Guo, G. Tao, X. Zhang, and D. Song, "BIRD: Generalizable backdoor detection and removal for deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 40786–40798, 2023.

[42] Y. Yu, S. Yan, and J. Liu, "A spatiotemporal stealthy backdoor attack against cooperative multi-agent deep reinforcement learning," in *2024 IEEE Global Communications Conference*, pp. 4280–4285, 2024.

[43] Y. Yu, J. Liu, S. Li, K. Huang, and X. Feng, "A temporal-pattern backdoor attack to deep reinforcement learning," in *IEEE Global Communications Conference*, pp. 2710–2715, 2022.

[44] S. Bradtke and M. Duff, "Reinforcement learning methods for continuous-time markov decision problems," *Advances in neural information processing systems*, vol. 7, 1994.

[45] K. Yu, C. Dong, L. Lin, and C. C. Loy, "Crafting a toolchain for image restoration by deep reinforcement learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2443–2452, 2018.

[46] V. J. Hodge, R. Hawkins, and R. Alexander, "Deep reinforcement learning for drone navigation using sensor data," *Neural Computing and Applications*, vol. 33, no. 6, pp. 2015–2033, 2021.

[47] C. Wang, J. Wang, Y. Shen, and X. Zhang, "Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2124–2136, 2019.

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv*, 2017.

[49] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[50] V. Mnih, A. P. Badia, *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[51] R. F. Woolson, "Wilcoxon signed-rank test," *Encyclopedia of biostatistics*, vol. 8, 2005.

[52] B. Justusson, "Median filtering: Statistical properties," *Two-dimensional digital signal prcessing II: transforms and median filters*, pp. 161–196, 2006.

[53] R. Kohavi and G. H. John, "The wrapper approach," in *Feature Extraction, Construction and Selection: a data mining perspective*, pp. 33–50, Springer, 1998.

[54] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[55] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "Openai gym," 2016.

[56] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning." http://pybullet.org, 2021.

[57] A. Raffin, "Rl baselines3 zoo." https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[58] K. Chen, S. Guo, T. Zhang, S. Li, and Y. Liu, "Temporal watermarks for deep reinforcement learning models," in *Proceedings of the 20th international conference on autonomous agents and multiagent systems*, pp. 314–322, 2021.

[59] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE symposium on security and privacy*, pp. 707–723, 2019.

[60] S. Bharti, X. Zhang, A. Singla, and J. Zhu, "Provable defense against backdoor policies in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14704–14714, 2022.

Appendix

A. Adaptive Freezing Mechanism

In Section 4.1, we introduced an adaptive freezing mechanism that automatically determines when the agent's policy has converged sufficiently before introducing backdoor triggers. The algorithm details are shown in Algorithm 2. The core of this mechanism is the Wilcoxon signed-rank test, a non-parametric hypothesis test that detects whether agent performance has plateaued, thereby reducing the risks of premature or delayed backdoor injection. We detail the procedure as follows:

Parameter Settings:

- T: **Performance evaluation frequency** (in time steps).
- M: **Episodes per evaluation** (e.g., 10), used to compute the average return for each evaluation.
- k: **Comparison window size** (e.g., $k = 5$). The most recent k evaluations are compared with the immediately preceding k evaluations.
- α: **Significance level**, typically set to 0.05, serving as the confidence threshold for determining significance.

Data Collection: At every evaluation interval T, we run M episodes, calculate the average reward p_i for the current evaluation, and maintain a sequence:

$$P = [p_1, p_2, \ldots, p_n]$$

Constructing Comparison Windows: Ensure at least $n \geq 2k$ evaluations. Define two sliding windows:

- **Recent Window:** $W_{\text{recent}} = [p_{n-k+1}, \ldots, p_n]$
- **Previous Window:** $W_{\text{previous}} = [p_{n-2k+1}, \ldots, p_{n-k}]$

Hypothesis Test (Wilcoxon Signed-Rank Test): We conduct a one-sided paired comparison to determine whether recent performance exceeds past performance. The hypotheses are:

$$\begin{aligned} \text{Null Hypothesis (H0):} \quad & \text{Median difference} \leq 0 \\ \text{Alternative Hypothesis (H1):} \quad & \text{Median difference} > 0 \end{aligned}$$

This asks: is the agent still improving, or has it plateaued?

Test Computation:

- **Paired Differences:** Compute $d_i = W_{\text{recent}}[i] - W_{\text{previous}}[i]$ for $i = 1, \ldots, k$.
- **Zero Differences:** Discard pairs where $d_i = 0$ (reduce k accordingly).
- **Ranking:** Rank absolute differences $|d_i|$ in ascending order (average ranks for ties).
- **Signed Ranks:** Assign ranks with the sign of d_i.
- **Sum of Ranks:** Compute W^+ (sum of positive signed ranks); this is the test statistic.
- **P-value:** For small k, use the exact distribution to compute the probability of observing a test statistic as extreme as W^+ under H_0; for large k, use the normal approximation with corrections.

Decision Rule: If the p-value $< \alpha$, reject H_0 and conclude that recent performance is still improving—continue training. If the p-value $\geq \alpha$, accept H_0 and declare performance converged: terminate the freezing period and proceed to trigger optimization.

Summary: The adaptive freezing mechanism provides a statistically principled, environment-agnostic criterion for policy convergence in deep RL. By employing the Wilcoxon signed-rank test on sliding windows of recent performance, we robustly detect convergence and minimize both premature and excessively delayed backdoor injection, thereby ensuring robust and reproducible attack effectiveness.

Algorithm 2: Adaptive Freezing Mechanism

Input: Evaluation interval T_{eval}; Window size k; Significance level α; Agent policy π
Output: Decision to end freeze period

Initialize performance history $\mathcal{P} \leftarrow \emptyset$; `is_frozen` $\leftarrow$ **true**; evaluation index $t \leftarrow 0$;
while `is_frozen` **do**
  Train agent policy π for T_{eval} steps;
  Evaluate agent on the primary task, obtain P_{t+1};
  Add P_{t+1} to $\mathcal{P}$;
  $t \leftarrow t + 1$;
  if $t \geq 2k$ ***and*** $t \bmod k = 0$ **then**
    $W_{\text{prev}} \leftarrow \{P_{t-2k+1}, \ldots, P_{t-k}\}$;
    $W_{\text{curr}} \leftarrow \{P_{t-k+1}, \ldots, P_t\}$;
    $p_{\text{value}} \leftarrow \text{WilcoxonTest}(W_{\text{curr}}, W_{\text{prev}})$;
    if $p_{value} > \alpha$ **then**
      `is_frozen` $\leftarrow$ **false**;
    end
  end
end
return End freeze period;

B. SHAP Value Calculation

This appendix provides a comprehensive description of the SHAP value calculation and trigger dimension selection process, supplementing the overview in Section 4.2.

B.1. Mathematical Foundation of SHAP Values

For a reinforcement learning agent with policy network π_θ (viewed as a function $f(s)$), our goal is to quantify the marginal contribution of each state dimension s_j in $s = (s_1, s_2, \ldots, s_D)$ to the model's output. The SHAP value $\phi_j(f, s)$ is formally defined as:

$$\phi_j(f, s) = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \big[f_x(S \cup \{j\}) - f_x(S) \big],$$

where F is the set of all features, and $f_x(S)$ denotes the expected model output when only features in S are known.

B.2. Practical SHAP Estimation via Local Linear Modeling

Direct computation of SHAP values is generally infeasible for nonlinear, high-dimensional RL policies. Thus, we use a local surrogate model inspired by LIME:

- **Background Dataset:** Construct a background data distribution $X_{\text{bg}} = \{s_{\text{bg}}^{(1)}, \ldots, s_{\text{bg}}^{(N_{\text{bg}})}\}$ by uniformly sampling environment states.
- **Explanation Set:** For N_{exp} target states $X_{\text{exp}} = \{s_{\text{exp}}^{(1)}, \ldots, s_{\text{exp}}^{(N_{\text{exp}})}\}$, compute feature contributions.
- **Coalition Sampling:** Each feature combination is represented by a binary vector $z' \in \{0, 1\}^D$. For $z'_j = 1$, feature j is present; for $z'_j = 0$, it is replaced by the mean $v_{\text{bg},j}$ of X_{bg}.
- **Perturbed Sample Generation:**

$$(s_{z'})_j = \begin{cases} s_j, & \text{if } z'_j = 1 \\ v_{\text{bg},j}, & \text{if } z'_j = 0 \end{cases}$$

  where s is the original state to be explained.
- **Local Surrogate Fitting:** Approximate f by a linear model in the neighborhood of s:

$$g(z') = \phi_0(s) + \sum_{j=1}^{D} \phi_j(s) z'_j,$$

  and minimize the weighted squared loss

$$\mathcal{L}(\{\phi_j(s)\}_{j=0}^{D}) = \sum_{z' \in \mathcal{Z}_{\text{sample}}} [f(s_{z'}) - g(z')]^2 \pi_s(z'),$$

  where the Shapley kernel is

$$\pi_s(z') = \frac{D - 1}{\binom{D}{k} k(D - k)}, \quad k = \sum_{j=1}^{D} z'_j,$$

with special care for $k = 0$ or $k = D$, where the denominator is set to a large value.

B.3. Trigger Dimension Ranking and Selection

For each $s_{\text{exp}}^{(i)}$, the above process yields a D-dimensional vector of SHAP values, $\Phi^{(i)} = (\phi_1(s_{\text{exp}}^{(i)}), \ldots, \phi_D(s_{\text{exp}}^{(i)}))$. The absolute value $|\phi_j(s_{\text{exp}}^{(i)})|$ reflects the contribution strength of feature j for the ith state.

To obtain a global ranking, aggregate over all N_{exp} instances:

$$I_j = \frac{1}{N_{\text{exp}}} \sum_{i=1}^{N_{\text{exp}}} |\phi_j(s_{\text{exp}}^{(i)})|,$$

and select the top-K dimensions with the highest I_j for trigger injection.

B.4. Practical Implementation Considerations

- **Coalition Sample Size:** To manage computation, M_{shap} is chosen in the range 10^2–10^4 per state.
- **Background Selection:** The use of pre-freeze or early-trajectory states as X_{bg} ensures distributional relevance and avoids bias.
- **Surrogate Model Fitting:** Weighted least squares (with the Shapley kernel) provides numerically stable SHAP value estimates.
- **Interpretability:** This methodology not only supports effective trigger dimension selection but also provides insights into the agent's policy sensitivity, valuable for both attack and defense research.

B.5. Summary

The above process enables context-sensitive, model-agnostic selection of highly influential state dimensions for trigger injection in RL backdoor attacks. By leveraging local linear SHAP approximations, we ensure that chosen dimensions have maximal impact on policy outputs, as validated by the experimental results in the main text.

C. Gradient-Based Trigger Value Computation

This appendix details the complete procedure for computing optimal trigger values via gradient-based search, complementing the main exposition in Section 4.3.

C.1. Algorithmic Implementation

The iterative process for trigger value optimization is given in Algorithm 3. For completeness, we clarify initialization, hyperparameters, and generalization to both discrete and continuous action settings.

Algorithm 3: Gradient-Based Trigger Value Optimization

Input: Trigger dimension p; Actor network π_θ; Feasible range $[v_{\min}, v_{\max}]$; Base state $\mathbf{s}_{\text{base}}$; Target action a_t (discrete) or $\mathbf{a}_t$ (continuous); Number of steps K; Learning rate schedule η_k; Momentum γ_{mom}; Action type $\mathcal{A}_{\text{type}}$; Log stability ϵ

Output: Optimized trigger value v^*

Initialize $v_0 \leftarrow (v_{\min} + v_{\max})/2$; $u_0 \leftarrow 0$;
for $k = 0$ **to** $K - 1$ **do**
  Construct $\mathbf{s}_k$ by setting $\mathbf{s}_{\text{base}}[p] \leftarrow v_k$;
  if $\mathcal{A}_{type}$ *is discrete* **then**
    $L_k \leftarrow -\log(\pi_\theta(a_t \mid \mathbf{s}_k) + \epsilon)$;
  else
    $L_k \leftarrow \|\boldsymbol{\mu}_\theta(\mathbf{s}_k) - \mathbf{a}_t\|_2^2$;
  $g_k \leftarrow \partial L_k / \partial v_k$;
  $u_{k+1} \leftarrow \gamma_{\text{mom}} u_k - \eta_k g_k$;
  $v_{k+1} \leftarrow \text{clip}(v_k + u_{k+1}, v_{\min}, v_{\max})$;
Set $v^* \leftarrow v_K$;
return v^*

C.2. Practical Considerations and Extensions

- **Base State Selection:** The base state $\mathbf{s}_{\text{base}}$ should be drawn from a representative distribution of clean states (e.g., after policy stabilization), ensuring the trigger context is realistic.
- **Learning Rate and Momentum:** The learning rate η_k may use decay or scheduled reduction for improved convergence. Momentum γ_{mom} helps smooth updates and escape local minima, with typical values between 0.5 and 0.9.
- **Clipping and Feasibility:** Enforcing $v_k \in [v_{\min}, v_{\max}]$ at each step guarantees that the trigger remains valid within the environment's observation constraints and avoids out-of-distribution artifacts.
- **Gradient Calculation:** For discrete action policies (e.g., categorical policies), gradients can be obtained via backpropagation through the log-probability of the target action; for continuous control, gradients are taken with respect to the squared error to the target action vector.
- **Stability and Termination:** The algorithm can terminate early if $|v_{k+1} - v_k|$ falls below a small threshold or if the loss L_k converges.
- **Multidimensional Triggers:** When attacking multiple dimensions simultaneously, the above method generalizes to vector-valued triggers $\mathbf{v}$; the update and gradient computations are performed jointly across all targeted dimensions.
- **Batch Optimization:** Optionally, multiple base states $\{\mathbf{s}_{\text{base}}^{(i)}\}$ can be used in parallel to optimize for an average-case effective trigger value across diverse initial conditions, further increasing robustness.

- **Target Action Selection:** For highest attack impact, the target action (or vector) may be selected adversarially, e.g., via maximizing disruption or minimizing reward, depending on the attack scenario.

C.3. Discussion

This gradient-driven optimization not only ensures that the computed trigger value is highly effective at activating the attacker's desired policy but also constrains the trigger within the agent's operational range, thus preserving stealth. The approach is flexible: it applies equally well to both discrete and continuous action spaces, and can be extended to support complex, context-sensitive triggers.

In summary, this procedure systematically exploits the differentiable structure of modern RL agents to discover trigger values that maximize backdoor efficacy without impairing benign performance, enabling precise and robust adversarial intervention.

D. Detailed Analysis of Defense Evasion

This appendix provides a more detailed discussion of the results presented in Section 6.6, concerning the resilience of `TooBadRL` attacks against standard defense mechanisms.

D.1. Evading Detection with Neural Cleanse

We utilized Neural Cleanse (NC) [59] to evaluate the detectability of our implanted backdoors. NC operates by reverse-engineering potential triggers through an optimization process that seeks minimal input perturbations capable of consistently forcing the model to a specific target output. When applied to PPO agents poisoned by `TooBadRL`, NC failed to identify any of the implanted triggers across all four benchmark environments (`Acrobot`, `BipedalWalker`, `Pendulum`, and `Hopper`), as summarized in Table 6.

This successful evasion can be attributed to several key characteristics of our attack methodology:

- **Minimal and Influential Triggers:** The SHAP-guided dimension selection and gradient-optimized trigger value ensure that the trigger is minimal while exploiting dimensions highly influential to the agent's policy. This can make the trigger perturbation less distinguishable from benign state variations that naturally occur, confounding detection methods that search for anomalous patterns.
- **Context-Sensitive Activation:** `TooBadRL` does not necessarily enforce a rigid, context-independent mapping from the trigger to a single malicious action across all possible states. The compromised policy's behavior, even when triggered, can retain a degree of context sensitivity. These factors likely render the trigger difficult for NC's optimization process to isolate as a distinct, universal pattern that works globally.

These results strongly suggest that `TooBadRL`'s optimized triggers can effectively evade detection by prominent backdoor identification methods like Neural Cleanse.

D.2. Resisting Elimination with RL-Sanitization

We further evaluated the robustness of `TooBadRL` attacks against trigger elimination defenses, specifically focusing on RL-Sanitization [60]. This defense aims to neutralize backdoor triggers by projecting state observations onto a lower-dimensional subspace, which is learned to preserve Normal Task Performance (*NTP*) while disrupting trigger activations. Following the methodology of prior work, we applied RL-Sanitization to PPO agents already poisoned by `TooBadRL` and subsequently assessed the change in the Balanced Utility Score (*BUS*).

As shown in Figure 7, the sanitization process had only a minimal impact on attack effectiveness. The *BUS* exhibited only marginal decreases: 2.6% in `Acrobot`, 5.8% in `BipedalWalker`, 1.1% in `Pendulum`, and 4.1% in `Hopper`. In all tested cases, the reduction in *BUS* remained below 6%, indicating that the attacks largely retained their effectiveness.

This resilience suggests that the meticulously optimized triggers are not easily nullified by the subspace projection employed by RL-Sanitization. The core features exploited by the trigger may lie within the principal components that are preserved by the sanitization technique itself, as these are often the same features necessary for maintaining high *NTP*. The targeted and minimal nature of the trigger makes it less likely to be filtered out as noise or a non-essential feature.