Institute of Computer Science IV
University of Bonn
Bachelor of Science in Computer Science

Comparing Defenses for Robust Reinforcement Learning against Adversarial Attacks in Tactical Networks

Bachelor's Thesis

Submitted by

Jonas Bode

Matriculation Number: 3374129

Email: jonas.bode@uni-bonn.de

Examiner: Prof. Dr. Meier[1] and Dr. Paulo H. L. Rettore[2]

Supervisor: Johannes Loevenich[2]

[1]Head of Institute Computer Science IV, University of Bonn, Germany
[1]Head of Institute, Fraunhofer FKIE, Wachtberg, Germany
Email: mm@cs.uni-bonn.de

[2]Research Scientists, Fraunhofer FKIE, Bad Godesberg, Germany
Email: [johannes.loevenich, paulo.lopes.rettore] @fkie.fraunhofer.de

In collaboration with the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), Bonn, Germany

February 14, 2023

Abstract

Reinforcement Learning (RL) is a promising approach for routing in highly dynamic environments such as Tactical Networks (TNs). However, the use of RL in TNs exposes a new vulnerability in the form of adversarial attacks, which is critical considering the high security requirements in military settings. To address this issue, we investigate the effectiveness of defensive methods that modify RL algorithms to enhance the robustness of trained agents against adversarial attacks. We compare a suite of such methods and evaluate their advantages and disadvantages in terms of robustness, training cost, and ease of implementation. Hence, our findings provide guidance for future researchers to develop Reinforcement Learning (RL) algorithms, which can fulfill the safety and security requirements for field deployment.

Acknowledgements

I would like to express my gratitude to Prof. Dr. Michael Meier and Dr. Paulo H. L. Rettore for serving as examiners for this thesis. Further, I would like to thank my supervisor at Fraunhofer FKIE Johannes Loevenich, whose work made this thesis possible.

Special thanks also belong to my fellow student researchers, Luca Liberto, Florian Spelter and Tobias Hürten. Our cooperation over the last year has been a font of inspiration, providing many insights along the way.

Lastly, I am deeply grateful for the continued support of my family.Their honest feedback, empathy, and love were essential in my research journey, and I could not have done it without them.

Contents

1 Introduction

Mobile communication devices such as mobile phones have enjoyed widespread adoption throughout society. Most of these devices rely on telecommunication towers or stationary routers. However, this necessary infrastructure does not exist everywhere nor is dependency upon static base stations desirable for users. Mobile Adhoc Networks (MANETs) promise a solution to this problem by offering a network consisting of multiple mobile nodes, each node acting as both sender and receiver. Yet, these dynamic conditions offered by MANETs also make effective routing extremely difficult. Not every node is capable of reaching all other nodes and ever-changing network conditions as well as node disconnections require a constant reevaluation of network paths. The problem is even more apparent for Tactical Networks (TNs) which utilize the capabilities of MANETs in a military context.

Therefore, TNs offer a challenging, yet also highly demanding, environment. Employed to connect various military assets and personnel, TNs need to be highly secure and reliable even though the networks operate in high-risk situations and can consist of many heterogeneous radio links connecting dynamically changing nodes. The integration of TNs into other communication systems (e.g. radio, satellite, vehicles, laptops, etc.) enables coverage of vast areas and complex organizational hierarchies vital in military operations such as disaster relief or crisis management. To ensure stability, the TN relies on each node to operate securely and reliably with known uncertainties.

Times of rising world tension highlight the need for new TN solutions and to further this cause research has begun to investigate the usage of machine learning to solve these problems. Traditional routing algorithms struggle to adapt to the dynamic environment of TNs with sufficient speed. This motivates attempts to utilize Reinforcement Learning (RL). RL may be uniquely suited to this problem, having proven itself invaluable in similar highly dynamic circumstances such as automated driving or robot control. Such a solution would even be applicable in a civilian context due to the increasing use of MANETs.

This motivated us to to develop experiments simulating TNs. We devised metrics for network performance leading to the formulation of an optimal control problem which can be solved by an RL agent [1–3]. This RL agent regulates network traffic by selecting the next node to which each IP datagram is forwarded. However, the same investigation [3] also found the agent to be vulnerable to adversarial attacks, enabling an adversary to manipulate the network behaviour.

While adversarial attacks are known to be effective against RL agents ever since 2017 [4], adversarial attacks had never before been considered in the context of TNs. Via attacks on the state space of the RL agent the authors showed that it is possible to lead the agent to faulty routing decisions, resulting in network delays and package loss [3]. If left unchecked, this new attack vector compromises the security and the reliability of the TN and has to be mitigated before such an approach could be employed in practice.

In response to this new threat, this thesis explores defensive techniques to safeguard the RL algorithm in TNs. We investigate a selection of defensive techniques against adversarial attacks by using the RL model of Loevenich et al. [3] and evaluating the effect of adversarial attacks against a defended version of the same model. Among the techniques we investigate are methods for robust learning [5], adversarial training [6, 7] and Policy Distillation

(PD) [8], allowing us to cover many different approaches promising resistance against adversarial attacks. The goal is to find a defensive technique capable of making adversarial attacks ineffective while raising the robustness of the RL agent to increase the reliability of the entire TN.

We discern the advantages and disadvantages of each defense method and identify the defense most well suited for our TN environment. Hence, we enable the future development of more reliable and secure RL agents inTNs and we indicate how future research might further improve these defenses.

In the following, Section 2 describes the current literature around RL solutions in TNs and adversarial attacks against RL. The necessary background information needed to contextualise this thesis is given in Section 3. Thereafter, Section 4 defines the problem statement for defending RL algorithms in TNs against adversarial attacks and approaches towards achieving this goal are provided in Section 5, with the respective results being discussed in Section 6. Lastly, Section 7 will summarize our contributions towards a new generation of secure and smart TNs and outline further avenues of research.

2 Related Works

The following discusses the research works upon which the rest of this thesis is based on. Further, it also includes context regarding the research of both adversarial attacks and RL solutions for TNs.

2.1 Attacks on Reinforcement Learning

Originating in the domain of image recognition [9], adversarial attacks have long since been known to be effective. Yet, the impact of adversarial attacks on RL did not become apparent until the experiments of Huang et al. [4] in 2017. Their results clearly showed the efficacy of adversarial attacks against RL algorithms. This subject has since been an expansive and active area of research, leading to increasingly more complex and more effective attacks. As a response, defenses to safeguard RL agents against such attacks have also been formulated. Many of these defenses also provide the added benefit of increasing the robustness of the RL agent, thus also increasing the ability of the agent to generalize.

A comprehensive review of both attacks and defenses for RL agents is given by Ilahi et al. [10]. In this work the authors propose a taxonomy for existing adversarial attacks and countermeasures, classifying attacks by evaluating the adversary's objective, knowledge, specificity and frequency. Further, they consider which space of the RL victim is attacked. Defenses are also categorized with respect to the most commonly used approaches.

Among defeneses, Vinitsky et al [7] propose Robustness via Adversary Populations (RAP). Due to a single adversary during training yielding insufficient results, they intend to increase victim robustness by having the target RL agent learn alongside a population of adversarial RL agents instead. During collection of trajectories, these adversarial agents learned online alongside the victim and try to sabotage the return by targeting the action-space of the victim.

Through this model robustness and generalization is increased as the victim learns to resist a multitude of different attack strategies.

Using their noise based adversarial attack Pattanaik et al. [6] also suggest a defensive method for robust training. During training they apply a so-called gradient attack, sampling noise in gradient direction to perturb the state.

Zhang et al. [5] also increase agent robustness during online adversarial training but instead of employing multiple adversarial agents targeting the action-space they choose to employ only a single adversarial agent targeting the observed state-space of the victim. For this they also use the conceptual framework of an State-Adversarial Markov Decision Process (SA-MDP) previously outlined by the same authors in [11]. The proposed Alternating Training with Learned Adversaries (ATLA) algorithm may also be utilized to allow the victim or the adversary to train for more steps than the other after new trajectories are collected, making precise control of the learning speed of both agent and adversary possible.

Another defensive approach is PD as demonstrated by Qu et al. [8]. They train a new neural network to replicate the behavior of the original RL agent victim, yet by using a new loss function their method called Adversary Agnostic Policy Distillation (A2PD) forces the new neural network policy to be highly robust to state adversarial attacks by increasing the prescription gap of the policy. Thereby the need for an adversarial agent is also eliminated.

2.2 Robust Tactical Network Solutions

The highly dynamic nature of MANETs as well as the steep requirements for reliability and security of TNs have always made routing challenging. Recently, research is increasingly looking for RL solutions to managing these IP data flows. In MANETs, Kaviani et al. [12] utilize multiple RL agents, one at each network node, to regulate datagram traffic by deciding whether to forward using broadcast or unicast, leading to the network being controlled by multi-agent RL. Due to the underlying Deep Q-Learning RL agents the authors named this routing algorithm *DeepCQ+*.

3 Foundations

To formalize the problem of defending TNs against adversarial attacks, a familiarity with the formalisms employed in RL and TN-related literature is necessary. This Section will also showcase Proximal Policy Optimization (PPO) [13], which is the RL algorithms used for our experiments, as well as the adversarial attacks used to evaluate robustness.

3.1 The Agent-Environment Duality in Reinforcement Learning

The lens of RL views problems as consisting of two components, the environment and an agent. At time t, the agent receives an observation s_t of the current environment state and chooses an action a_t which is then executed in the environment leading to a reward r_t and a new observation s_{t+1} before this cycle repeats until the environment terminates. The observed state s_t is the only information the agent receives at each time step and thus the agent uses this information to determine the next action a_t. This usually finite chain leading form the initial state s_0 at the start of the environment till its termination at time T is also called episode, trajectory or rollout τ and can be expressed as

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T). \tag{1}$$

More formally, this is assumed to be a Markov Decision Process (MDP) and the problem can be written as a 5-tuple $(\mathcal{S}, \mathcal{A}, R, P, p_0)$. In this tuple, $\mathcal{S}$ is the state space of all possible environment states $s \in \mathcal{S}$ which are then observed by the agent and $\mathcal{A}$ is the action space consisting of all possible actions $a \in \mathcal{A}$ the agent can choose from at any time.

$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. Using this function, at a time t the reward r_t is therefore given as $r_t = R(s_t, a_t, s_{t+1})$. The reward function acts as a metric for the performance of the agent, i.e. a high reward indicates a good outcome.

Lastly, the environment is encapsulated in the transition function $P : \mathcal{S} \times \mathcal{A} \to (\mathcal{P}(\mathcal{S}))$. This transition function takes current state and action as input and outputs a probability distribution resulting in $P(s_{t+1}|s_t, a_t)$ being the probability that the next state is s_t given the current state s_t and the chosen action a_t. Since there is no previous state at time $t = 0$ the beginning state is determined by the starting state distribution p_0.

An RL training algorithm thus need to train the agent which encapsulates a policy function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. For this function $\pi(a_t|s_t)$ is the probability of action a_t being chosen as the next action if state s_t is observed by the agent during training. It is to note that after training the stochastic policy is usually converted to a deterministic policy by choosing the action with the highest probability for each observation. This same policy is employed at every time step during a single episode, which is sufficient due to the Markov property stating

$$\pi(a_t|s_t) = \pi(a_t|s_0, s_1, \ldots, s_t). \tag{2}$$

Using all of the above enables the sampling of episodes according to a policy, $\tau \sim P(\tau|\pi)$ or using even simpler notation $\tau \sim \pi$. This duality of agent and environment is also expressed in Figure 1 where the cyclic nature becomes evident and it is visible how each time step leads to the action, state and reward in the next time step.

As a good agent should provide us with an optimal policy instead of a random one, we utilize the return

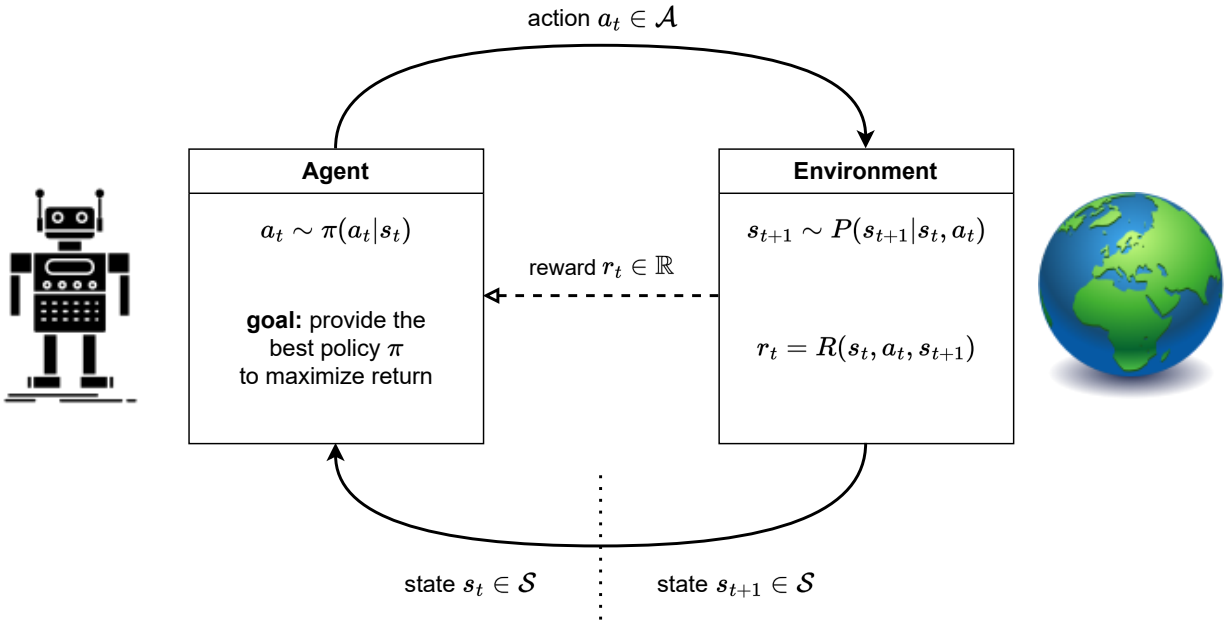


Figure 1. Agent-Environment duality. During each time step the agent receives state observation s_t and uses this to select action a_t. This action is then executed in the environment leading to the reward r_t and the next state observation s_{t+1}.

$$R(\tau) = \sum_{t=0}^{T-1} r_t \tag{3}$$

to evaluate a policy's performance on an episode. However, to incentivise short term gains over equal long term gains, using a discount factor $\gamma \in [0, 1]$ this return is modified into the discounted return

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t. \tag{4}$$

This discounting factor γ can also be included in the tuple $(\mathcal{S}, \mathcal{A}, R, P, p_0, \gamma)$. The ultimate goal of an RL agent to choose an optimal policy π^* to maximise the expected value of the discounted return can hence be formalized as

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi} [R(\tau)] \tag{5}$$

To find the optimal policy, it is often useful to consider the value of a state or of a state-action pair for any policy. We can find these through the use of the value function $V^\pi(s)$ and the action-value function $Q^\pi(s, a)$ respectively, both evaluating the expected return of an episode starting with the function arguments. More explicitly they can be written as

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi} [R(\tau)|s_0 = s] \tag{6}$$

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi} [R(\tau)|s_0 = s, a_0 = a]. \tag{7}$$

For a fixed state s we can also look at the difference between the value function and the action-value function for an action a. This measures how much benefit the policy gains by choosing action a in state s and is called the advantage function $A^\pi(s,a)$,

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{8}$$

3.2 Policy Gradient & Proximal Policy Optimization

To find the optimal policy π^* we can use deep learning techniques by representing the agent as a neural network, which takes a state s as input returns a probability distribution over all actions. Thus the agent's policy π_θ is controlled by the neural network parameters θ. Reformulating the agents goal as a function using Equation 5 we can build the reward function

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] \tag{9}$$

as a function of the parameterized policy π_θ which we seek to maximise. If we know the policy gradient$\nabla_\theta J(\pi_\theta)$, i.e. the gradient of the reward function w.r.t. the network parameters θ defining the agents policy, we can accomplish this by using stochastic gradient ascent with the iterative update at time step k of the form

$$\theta_{k+1} = \theta + \eta\nabla_\theta J(\pi_\theta)|_{\theta_k}. \tag{10}$$

The policy gradient $\nabla_\theta J(\pi_\theta)$ can be estimated via Monte-Carlo methods, first suggested by Williams [14], similarly to the gradient of a usual deep learning loss function by

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] && (11)\\
&= \nabla_\theta \int_\tau P(\tau|\pi_\theta)R(\tau) && (12)\\
&= \int_\tau P(\tau|\pi_\theta)\nabla_\theta \log(P(\tau|\pi_\theta))R(\tau) && (13)\\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log(P(\tau|\pi_\theta))R(\tau)] && (14)\\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t))R(\tau)\right]. && (15)
\end{aligned}$$

The thus established iterative updates can be improved through the introduction of a critic network which is trained to approximate the value function. The critic V_ω with parameters ω is therefore trained to fulfill $V_\omega(s) = V^\pi(s)$ This enables the use of advantage estimation [15]

by substituting the value function with the critic and results in Actor-Critic Algorithms proposed by Konda et al. [16]. In these algorithms the policy network is also called actor leading to the their name. The estimated advantage may then replace the empirical return in Equation 15, reducing variance for better learning.

Schulman et al. [13] used this framework to develop the PPO algorithm, which utilizes the concept of a trust region [17], reducing implementation complexity and computational burden. PPO [13] uses the surrogate objective function

$$L^{\text{CLIP}} := \mathop{\mathbb{E}}_{(s_t,a_t,\hat{A}_t)\sim\pi_\theta} \left[\sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \right) \hat{A}_t \right] \tag{16}$$

to replace the negative of the reward function Equation 9 and can be optimized using Stochastic Gradient Descent (SGD) methods such as Adam [18]. This surrogate objective uses hyperparameter ϵ and the estimated advantage $\hat{A}_t$. It can also be computed offline, after samples have been collected, using the saved probability $\pi_{\theta_{\text{old}}}(a_t|s_t)$ calculated according to the network parameters θ_{old} from the time of sample collection.

Algorithm 1 Basic PPO algorithm used in this thesis from Schulman et al. [13]. Critic omitted for readability.

Require: hyperparameters N_{iters}, T, N_{epochs} and environment *env*
  Initialize policy actor network π_θ and with parameters θ
  for $\text{epoch} = 1, \ldots, N_{\text{epochs}}$ **do**
    $\theta_{\text{old}} \leftarrow \theta$
    $\mathcal{D} \leftarrow \{\}$ ▷ Initialize episode buffer
    for $t = 1, \ldots, T$ **do** ▷ Collect T time steps worth of data
      if *env* has terminated **then**
        reset *env* and obtain new s_t
      end if
      Sample $a_t \sim \pi_{\theta_{\text{old}}}(a_t|s_t)$
      execute a_t in *env* and obtain r_t, s_{t+1}
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t, r_t, \pi_{\theta_{\text{old}}}(a_t|s_t)\}$
    end for
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ using critic
    for $\text{iter} = 1, \ldots N_{\text{iters}}$ **do** ▷ Train agent using collected data
      Sample mini-batch from $\mathcal{D}$
      Optimize L^{CLIP} w.r.t. θ using mini-batch and SGD
    end for
  end for

Algorithm 1 shows the resulting RL scheme which is utilized as the main algorithm for this thesis. Each iteration consists of episode collection using the environment before this data is used to optimize the policy actor π_θ and the critic V_ω for multiple epochs.

3.3 Adversarial Attacks

While RL has been demonstrated to be capable of solving complex dynamic control tasks, the security of the agent's decision making is not assured and needs to be investigated. Adversarial attacks aim to reduce the agent's performance via malicious perturbations, making them highly dangerous in areas with high security demands.

Since the work of Szegedy et al. [9] in 2014 adversarial attacks have become a prominent field of investigation in deep learning algorithms for image classification. Huang et al. [4] applied similar methods and found that RL algorithms also exhibit vulnerabilities to adversarial attacks.

The goal of an adversarial attack is to create a modified state observation $s' = s + \delta$ using a perturbation δ, such that $\pi(a|s')$ leads to the minimal return in the environment. As an unlimited perturbation could change any state into any other state, the perturbation δ is regulated using a threshold ϵ and a norm such that $\|\delta\| \leq \epsilon$. This also makes the attack more realistic since an ideal attack would be as stealthy as possible, favouring lower perturbation values and thereby making the attack harder to detect.

Two such attacks are showcased in Figure 2 against an agent optimized to play Pong. Both attacks lead to a wrong decision by the agent, causing the agent to miss the ball and lowering the return. Depending on the norm and the threshold ϵ used to regulate the perturbation's intensity, an adversarial attack may be imperceptible to humans.

In addition to the norm constraint limiting the extent of the perturbation, an adversarial attack may also choose to not attack during every time step of an episode, further limiting detection. Hence an adversary executes two steps as depicted in Figure 3. First, the adversary chooses if a given state should be attacked. Second, the adversary crafts the adversarial perturbation used to create the adversarial state. Hence, an adversary consists of two components, the attack timing and the attack method.

Loevenich et al. [3] found tactical networks to be vulnerable to adversarial attacks. They discovered that attacks utilizing more knowledge about the victim agent proved to be more effective. For the purpose of this thesis, we will test the defensive methods against the same selection of adversarial attacks as was used by Loevenich et al. [3]. This selection is comprised of attack timings and attack methods using varying levels of complexity and knowledge.

3.3.1 Attack Timings

The simplest attack timing is the **uniform attack** which was first employed by Huang et al. [4] but first named such by Lin et al. [19]. This timing attacks in every time step making it both the least stealthy timing as well as the most powerful with regards to reducing the victims return.

Lin et al. [19] also propose a more advanced method. To evaluate if a state is worth attacking the authors utilize a preference function $c : \mathcal{S} \to \mathbb{R}$. This function is a measure of how important the choice of a specific action is for the victim agent. As such it is given as the difference between the most and the least likely actions as determined by the victim's policy

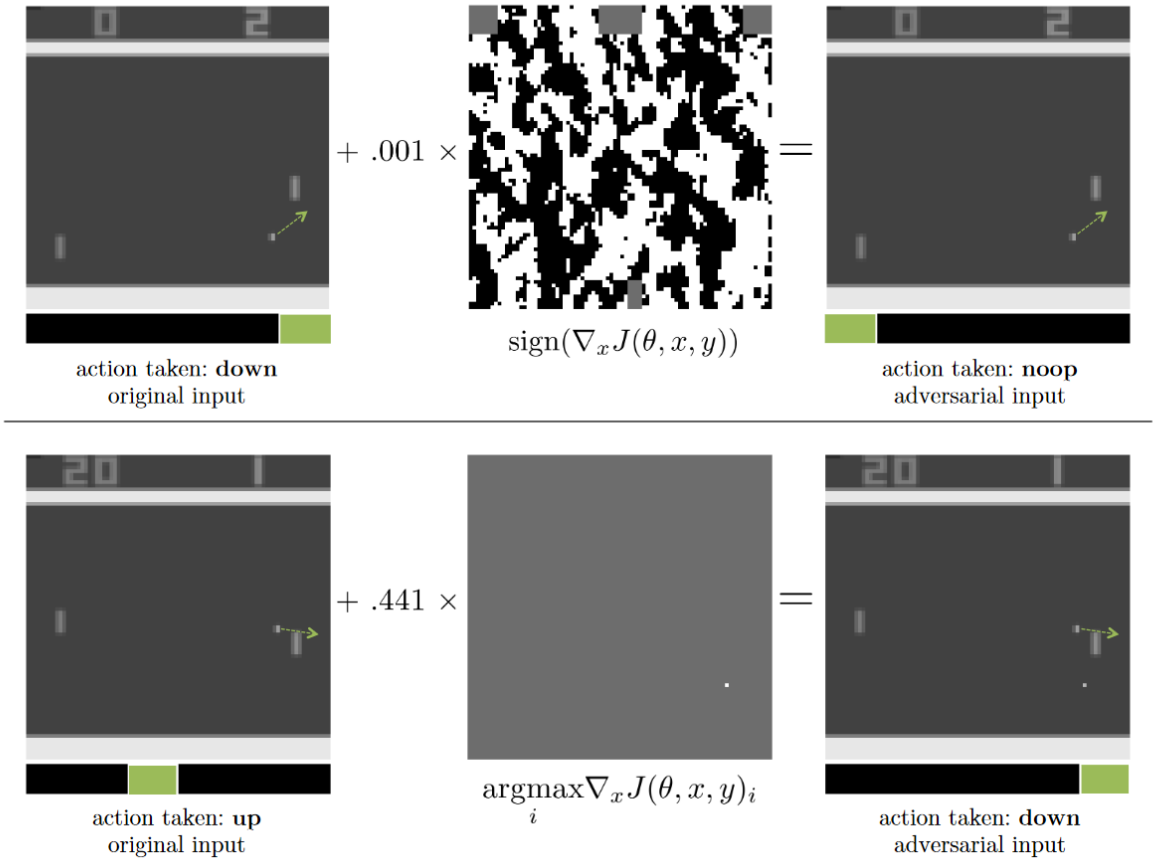


Figure 2. Figure from Huang et al. [4] displaying an example of an adversarial attack against an RL agent playing Pong. Both are computed using an Fast Gradient Sign Method (FGSM) method but use different norms. The top attack perturbs each pixel using an L_∞-norm, leading to a wrong action being selected even though the difference is imperceptible for humans. The bottom attack perturbs only a small number of pixels by a large amount using an L_1-norm, thus creating a fake ball. In both cases the attack causes the agent to miss the ball.

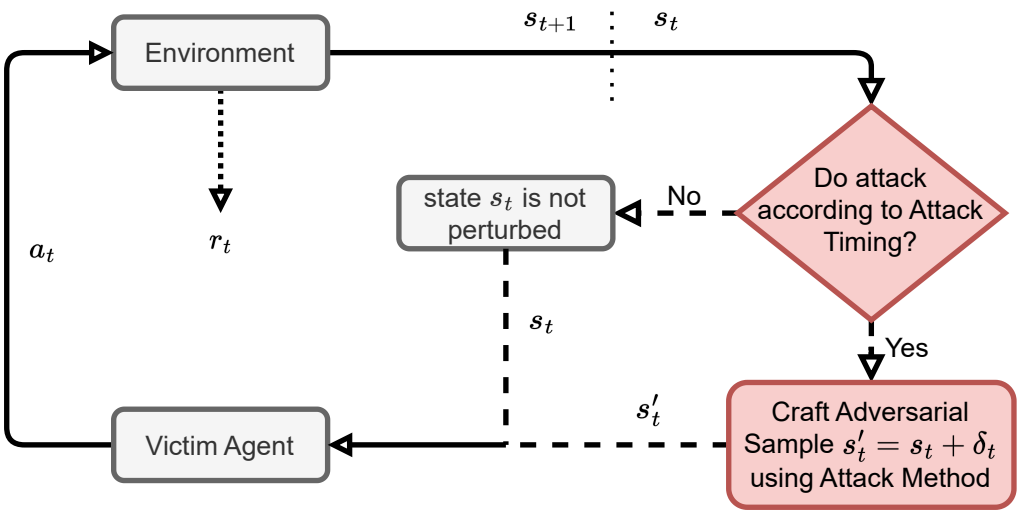


Figure 3. Attack methodology. Steps controlled by adversary colored in red.

$$c(s) = (\max_a \pi(a|s)) - (\min_a \pi(a|s)). \tag{17}$$

When the victim agent makes an important action choice in state s, it should strongly prefer the most advantageous action while avoiding the worst action. Thus, such a state will have a high preference $c(s)$. The **strategically-timed attack** [19] uses a threshold $\beta \in [0, 1]$ and will only attack a given state s should the preference exceed this threshold

$$c(s) \geq \beta. \tag{18}$$

Thus β regulates the attack frequency and if properly chosen should enable the adversary to restrict itself to only attacking the most important states. The higher the value of β the less frequent are attacks. Note that for $\beta = 0$ this timing is equivalent to the uniform attack.

3.3.2 Attack Methods

Both of these attack timings can be paired with different attack methods. As many attack methods have been proposed in recent years, we only focus on a selection of the most popular ones in this thesis. To craft the best adversarial sample these methods need to create a perturbation δ which satisfies the constraint in Equation 18.

In this thesis we use an L_∞-norm for this constraint. In a tactical network this will make the adversarial harder to attack as the minor perturbations for each value will be harder to detect than large perturbations for few values as background noise would impede the detection process.

The first attack method used in this thesis is the **naive attack**, first suggested by Pattanaik et al. [6]. The naive attack searches the space of all admissible perturbations by iterative sampling of noise.

Algorithm 2 Naive attack method used to create adversarial perturbation based on Pattanaik et al. [6]. We use $\alpha = 0.5$, $\beta = 0.5$ for all experiments.

Require: state s, hyperparameters n, α, β, ϵ and victim policy π
  $a \leftarrow \arg\max_a \pi(a|s)$
  $p \leftarrow \pi(a|s)$
  $\delta \leftarrow 0$
  for i $= 1, \dots n$ **do**
    Sample $\delta_i \sim \epsilon \cdot (2 \cdot \text{Beta}(\alpha, \beta) - 1)$ ▷ sample perturbation
    $p_i \leftarrow \pi(a|s + \delta_i)$ ▷ evaluate perturbation suing policy
    if $p > p_i$ **then** ▷ decide if new perturbation is better than previous ones
      $\delta \leftarrow \delta_i$
      $p \leftarrow p_i$
    end if
  end for
  return $s' \leftarrow s + \delta$ ▷ Return adversarial state

This naive attack is given in Algorithm 2. For n iterations noise is sampled from a beta distribution. Consecutively, the resulting noise is scaled and shifted to be in the interval of $[-1, 1]$ before being scaled once more using ϵ to fully utilize the available perturbation space.

The parameters for the beta distribution used by Pattanaik et al. [6] were $\alpha = 1$, $\beta = 1$ but for this thesis we use $\alpha = 0.5$, $\beta = 0.5$. This choice increases the probability of each perturbation value being close to $\pm\epsilon$ while the added scaling and shifting still ensures a 0 mean for the perturbation.

The generated perturbations are then evaluated using the victim policy π. Ultimately the algorithm returns the adversarial sample δ which decreases the probability of the original chosen action a the most. Hence, this adversarial sample also maximizes the probability of a different action being chosen.

This attack can be further improved by applying the perturbation in the direction which causes the greatest loss in return. While the naive attack uses a random direction the **gradient attack** of Pattanaik et al. [6] refines this process by applying the perturbation in the direction of the gradient of a specialized objective function, thus searching the admissible perturbations more effectively.

To calculate the gradient we use an objective function $J'(s, \pi)$ dependent on the current state s and the victim policy π. The authors hence suggest the cross-entropy loss between a hypothetical policy always choosing the worst action and the victim policy's distribution $\pi(a|s)$, resulting in

$$P'_{\pi,s}(a) = \begin{cases} 1 & a = \arg\min_{a'} \pi(a'|s) \\ 0 & \text{else} \end{cases} \tag{19}$$

$$J'(s, \pi) = -\sum_{a \in \mathcal{A}} P'_{\pi,s}(a) \log(\pi(a|s)). \tag{20}$$

Algorithm 3 Gradient attack method used to create adversarial perturbation based on Pattanaik et al. [6]. We use $\alpha = 0.5$, $\beta = 0.5$ for all experiments.

Require: state s, hyperparameters n, α, β, ϵ and victim policy π
  $a \leftarrow \arg\max_a \pi(a|s)$
  $p \leftarrow \pi(a|s)$
  $\delta \leftarrow 0$
  $\text{grad} \leftarrow \frac{\nabla_s J(s,\pi)}{\|\nabla_s J(s,\pi)\|_2}$ ▷ determine gradient direction to ensure constraint
  for i $= 1, \dots n$ **do**
    Sample $\delta_i \sim -\text{grad} \cdot \epsilon \cdot |(2 \cdot \text{Beta}(\alpha, \beta) - 1)|$ ▷ sample perturbation using gradient
    $p_i \leftarrow \pi(a|s + \delta_i)$ ▷ evaluate perturbation suing policy
    if $p > p_i$ **then** ▷ decide if new perturbation is better than previous ones
      $\delta \leftarrow \delta_i$
      $p \leftarrow p_i$
    end if
  end for
  return $s' \leftarrow s + \delta$ ▷ Return adversarial state

We can now use the gradient $\nabla_s J'(s, \pi)$ to modify the naive attack resulting in Algorithm 3 for the gradient attack. To ensure we do not violate the ϵ constraint we normalize $\nabla_s J'(s, \pi)$ using an L_2-norm.

The last attack method this thesis uses is the **adversarial policy attack**. This attack trains an RL agent to construct the necessary adversarial samples.

Given a victim agent which solves the RL problem $W_v = (\mathcal{S}, \mathcal{A}, R, P, p_0, \gamma)$ as specified in Section 3.1, we can define a second RL problem $W_a = (\mathcal{S}, \mathcal{A}', -R, P', p_0, \gamma)$ which is derived from W_v. In W_a the reward function from W_v is negated and $\mathcal{A}'$ is the space of all perturbations satisfying the ϵ constraint. To define P' we need a policy $\pi^{(v)}$ intended to solve RL problem W_v as well as $\delta_t \in \mathcal{A}'$ allowing us to formulate

$$a_t \sim \pi^{(v)}(a_t|s_t + \delta_t) \tag{21}$$

$$P'(s_{t+1}|s_t, \delta_t) = P(s_{t+1}|s_t, a_t). \tag{22}$$

Hence, if we can find a policy $\pi^{(a)}$ solving RL problem W_a we can use this adversarial policy as an attack method. In this case $\delta \sim \pi^{(a)}(\delta|s)$ is used to get the perturbation needed to craft the adversarial sample. Since we have formulated W_a, we can find such an adversarial policy through the use of existing RL algorithms such as the already introduced PPO [13].

Note that all of the adversarial attacks used in this thesis require knowledge about the victim agent. The adversarial policy requires the victim policy as an oracle and the environment the victim agent was trained on, but needs these only during training. Once trained the adversarial policy can be executed as a black-box attack. The naive and the gradient attack method require an oracle giving the victim policy during the execution of the attack. Further, the gradient method also needs the policy to be solvable using backpropagation.

A benefit of the adversarial policy method is computation time. While the naive and the gradient attack need to pass noise through the oracle for n iterations, the adversarial policy needs only to be evaluated once to craft the adversarial sample. As speed may be a deciding factor in practical usage of adversarial attacks this speed up is of great benefit to the attacker.

3.4 Tactical Network Environment

To employ RL in TNs we need to formulate an environment, including a reward function. For this purpose, this thesis will use the TN environment presented by Loevenich et al. [3].

The authors presuppose that a tactical network can be modeled by a graph $G = (V, E)$. This graph includes both the tactical network nodes V and the weighted edges E representing the network links. At the start of the environment, the TN is generated as a random Barabási–Albert (BA) graph.

The TN constraints are modeled by weights $w : E \to \mathbb{R}^3$ which map each network link $e = (v_i, v_j) \in E$ to a tuple $w(e) = (q_e, l_e, d_e)$, signifying the queue length $q_e \in [0, 1]$, the packet loss rate $l_e \in [0, 1]$ and the link data rate changes $d_e \in [0, 10]$ respectively.

Using this graph model, any path ϕ of a single packet traveling through the network from source $S \in V$ to target $T \in V$ is defined as a set $\phi \subseteq E$. Hence, for any path we can compute a cost as the weighted sum of the overall queue length, packet loss rate and data rate changes using

$$w(\phi) = \omega_1 \sum_{e \in \phi} q_e + \omega_2 \left(1 - \prod_{e \in \phi} (1 - l_e) \right) + \omega_3 \sigma^2[d_e]. \tag{23}$$

This path cost utilizes three importance weights ω_1, ω_2, ω_3 as well as an estimate of the uncertainty within the network, which results from the prepossessing of the dynamic link conditions as described in Loevenich et al. [20]. With Equation 23 finding an optimal path ϕ^*, which is the goal of the RL agent, can also be stated as an optimal control problem

$$\begin{aligned} &\min \omega_1 \sum_{e \in \phi} q_e + \omega_2 \left(1 - \prod_{e \in \phi} (1 - l_e) \right) + \omega_3 \cdot \sigma^2[d_e] \\ &\text{subject to:} \\ &\forall e \in \phi \; q_e \leq 1.0, \; 1 - \prod_{e \in \phi} 1 - l_e \leq 1.0, \; 0.0 \leq d_e \leq 9.6. \end{aligned} \tag{24}$$

The thus formulated goal of the agent is notably always achievable since a generated BA graph is connected per definition, guaranteeing the existence of a path between any source and target node. After graph generation the source and target will be determined by a uniform selection over all nodes.

We define the action space $\mathcal{A}$ for the agent to consist of all edges connecting the current node to any other direct neighbor. This enables the agent to control the path of the packet. Should the agent attempt to choose any node that is not a direct neighbor it will be heavily penalized as such an action will result in immediate packet loss.

Similarly the state observation $s \in \mathcal{S}$ in each time step includes the network link parameters q_e, l_e, d_e for each edge connecting the current node v_i to any possible next node. Further

s includes a binary encoding, detailing which nodes are the direct neighbors of the current node.

Lastly, we define the reward function R. We wish to penalize the agent for taking an illegal action, give out a large reward for a packet successfully reaching the target node T and after any legal action which moves to an intermediary node we will give out a reward based on a function emphasizing the path performance according to the optimal control problem 24. Since the reward function only depends on the current node v_i and the node v_j resulting from the agent choosing to traverse the link $e = (v_i, v_j)$, we may simplify the notation for reward function by writing $R(e)$ instead of $R(s_t, a_t, s_{t+1})$. Thus, we get

$$R(e = (v_i, v_j)) = \begin{cases} 100 & v_j = T \\ -5 & e \text{ illegal action} \\ \frac{1}{w_1} \cdot q_e + \frac{1}{w_2} \cdot (1 - (1 - l_{e_{v_i}}) \cdot (1 - l_e)) + \frac{1}{w_3} \cdot \mathbb{E}[d_e] & \text{else.} \end{cases} \tag{25}$$

$$= r_t \tag{26}$$

4 Problem

Based on the environment definition in Section 3, it is evident that adversarial attacks aimed at reducing the agent's reward will result in degraded network performance. These attacks induce the selection of sub-optimal paths, resulting in higher costs or lost packets. Given the non-negligible impact of adversarial attacks demonstrated by Loevenich et al. [3], it is imperative to defend the RL algorithm.

In order to mitigate the impact of adversarial attacks, it is necessary to enhance the robustness of the policy, allowing the agent to maintain a level of performance that is comparable to its default behavior, i.e., the performance achieved when it is not under attack. To achieve this, we propose modifying the RL algorithm that generates the agent's policy π.

To provide a more formal description of the problem at hand, we build upon the foundational elements presented in Section 3. Thus, the primary objective of this thesis is to resolve the following problem.

Problem 4.1 *Given a graph environment representing a TN as described in Section 3.4 and an RL algorithm computing a policy π, is it possible to change the RL algorithm to make the resulting policy π more robust against adversarial attacks without significantly decreasing policy default performance?*

If it is possible to increase the robustness of the policy without significantly decreasing the default performance, the second goal of this thesis is to evaluate the advantages and disadvantages of the proposed solutions. Key characteristics to consider include the *additional training time required*, *the ease of deployment*, *adaptability to new RL approaches*, and any *potential reduction* in default performance. Thus, we aim to select the defensive method most suitable to be used in future TN development.

Problem 4.2 *Given the existence of multiple solutions to Problem 4.1, which solution(s) are the most appropriate for use in the development of TNs, and what are their respective advantages and disadvantages?*

Having formalized the problem statements we aim to adress, the following Section 5 will focus on the different defensive solutions we are evaluating.

5 Defense Methodology

Achieving more robust RL requires modification of the policy network. This becomes possible through alteration of the learning algorithm or by transforming the policy network after the RL agent has been trained. This task is difficult due to the need to retain the return achieved by a less robust agent.

Further, these modifications often lead to an increase in computation cost per epoch. This leads to the modified RL algorithms achieving less training given the same time or requiring more time to achieve the same amount of training. For our experiments we chose to give each agent the same amount of training steps even if this caused the training to last a lot longer for certain versions.

This Section will explain the schemes we examine. Each defensive method provides different advantages and many can also be combined with each other.

5.1 Robust Training using Gradient Attacks

To defend an algorithm against adversarial attacks, many strategies choose to attack the agent during training. Doing so forces the agent to learn how to achieve a high return while subject to an adversarial attack. After training this attack is removed for deployment. The agent retaining its learned robustness is hence more capable of negating adversarial attacks.

Using the gradient attack proposed in the same work, Pattanaik et al. [6] developed a training scheme aiming to increase robustness through adversarial training. For this thesis we will employ a modified version resulting in Algorithm 4 which we name Robust Training using Gradient Attacks (RTGA).

Our version differs in two main aspects. First, RTGA adapts the work of Pattanaik et al. [6] to utilize a PPO [13] agent. Second, while originally an agent was first trained normally before the gradient attack was enabled, our version applies the gradient attack even at the start of training.

Algorithm 4 shows the resulting RTGA. This algorithm modifies the standard PPO agent as seen in Algorithm 1 during episode collection. Before using the policy network π_θ to determine the chosen action, an adversarial perturbation is added according to the gradient attack Algorithm 3 to create an adversarial state s'_t. This adversarial state s'_t is then treated as if it was the agent's only observation.

Algorithm 4 RTGA integrated into the PPO version given in Algorithm 1. Modifies the robust training [6] for PPO.

Require: hyperparameters N_{iters}, T, N_{epochs} and environment *env*
  Initialize policy actor network π_θ and with parameters θ
  for $\text{epoch} = 1, \ldots, N_{\text{epochs}}$ **do**
    $\theta_{\text{old}} \leftarrow \theta$
    $\mathcal{D} \leftarrow \{\}$ ▷ Initialize episode buffer
    for $t = 1, \ldots, T$ **do** ▷ Collect T time steps of data
      if *env* has terminated **then**
        reset *env* and obtain new s_t
      end if
      create $s't$ using gradient attack (see Algorithm 3) ▷ craft adversarial state
      Sample $a_t \sim \pi_{\theta_{\text{old}}}(a_t|s'_t)$
      execute a_t in *env* and obtain r_t, s_{t+1}
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{s'_t, a_t, r_t, \pi_{\theta_{\text{old}}}(a_t|s_t)\}$
    end for
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ using critic
    for $\text{iter} = 1, \ldots N_{\text{iters}}$ **do** ▷ Train agent using collected data
      Sample mini-batch from $\mathcal{D}$
      Optimize L^{CLIP} wrt. θ using mini-batch and SGD
    end for
  end for

5.2 Alternating Training with Learned Adversaries

Zhang et al. [5] also suggest adversarial training but use an adversarial policy instead of a gradient attack. This adversary also uses PPO and is trained together with the agent.

ATLA is detailed in Algorithm 5. This algorithm uses two RL agents. The target agent given by policy $\pi^{(v)}$ will be used after training to process state observations s and choose actions a. The adversarial policy agent $\pi^{(a)}$ is used to create adversarial perturbations.

Each epoch begins by keeping the adversary $\pi^{(a)}$ static, collecting data and training the target agent $\pi^{(v)}$. After this the training switches. Now the target agent $\pi^{(v)}$ remains static while data collection and training is done for the adversary $\pi^{(a)}$. A single epoch is only finished after both agents have been trained.

This approach allows for the adversary to adapt to the agents learned behavior, enabling the continuous adversarial policy attack to shift which target agent behavior to exploit. Hence the target agent is forced to avoid developing intrinsic weaknesses as the adversarial policy will learn to exploit these.

5.3 Robustness via Adversarial Populations

RAP, originating in the work of Vinitsky et al. [7], similarly uses learned adversaries during training. Instead of a single state adversarial agent used in ATLA, RAP uses multiple action

Algorithm 5 ATLA [5] integrated into the PPO version given in Algorithm 1. Both the adversary and the target are PPO agents.

Require: hyperparameters $N^{(v)}_{\text{iters}}$, $N^{(a)}_{\text{iters}}$, T, N_{epochs} and environment *env*
  Initialize policy actor network $\pi^{(v)}_{\theta}$ and with parameters θ
  Initialize adversarial policy network $\pi^{(a)}_{\omega}$ and with parameters ω
  for $\text{epoch} = 1, \ldots, N_{\text{epoch}}$ **do**
    $\theta_{\text{old}} \leftarrow \theta$
    $\mathcal{D} \leftarrow \{\}$ ▷ Initialize episode buffer
    for $t = 1, \ldots, T$ **do** ▷ Collect T time steps of data
      if *env* has terminated **then**
        reset *env* and obtain new s_t
      end if
      $s'_t \leftarrow s + \arg\max_{\delta} \pi^{(a)}_{\omega}(\delta|s_t)$ ▷ create adversarial sample using static pi^{a}_{ω}
      Sample $a_t \sim \pi^{(v)}_{\theta_{\text{old}}}(a_t|s'_t)$
      execute a_t in *env* and obtain r_t, s_{t+1}
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{s'_t, a_t, r_t, \pi^{(v)}_{\theta_{\text{old}}}(a_t|s'_t)\}$
    end for
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ using critic
    for $\text{iter} = 1, \ldots N^{(v)}_{\text{iters}}$ **do** ▷ Train target using collected data for target
      Sample mini-batch from $\mathcal{D}$
      Optimize L^{CLIP} w.r.t. θ using mini-batch and SGD
    end for

    $\omega_{\text{old}} \leftarrow \omega$
    $\mathcal{D} \leftarrow \{\}$ ▷ Initialize episode buffer
    for $t = 1, \ldots, T$ **do** ▷ Collect T time steps of data for adversary
      if *env* has terminated **then**
        reset *env* and obtain new s_t
      end if
      Sample $\delta_t \sim \pi^{(a)}_{\omega_{\text{old}}}(\delta|s_t)$
      $a_t \leftarrow \arg\max_a \pi^{(v)}_{\theta}(a|s_t + \delta_t)$ ▷ calculate action using static pi^{v}_{θ}
      execute a_t in *env* and obtain r_t, s_{t+1}
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, \delta_t, -r_t, \pi^{(a)}_{\theta_{\text{old}}}(\delta_t|s_t)\}$
    end for
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ using critic
    for $\text{iter} = 1, \ldots N^{(a)}_{\text{iters}}$ **do** ▷ Train adversary using collected data
      Sample mini-batch from $\mathcal{D}$
      Optimize L^{CLIP} w.r.t. ω using mini-batch and SGD
    end for
  end for

adversarial agents. An action adversarial attack does not perturb the observed state but instead affects the action space.

While in its original form devised to be used in continuous control problems, for this thesis we have adapted RAP to discrete action spaces. Given the probability distribution of the target policy $\pi^{(v)}(\cdot|s_t)$ and the adversary policy $\pi^{(a)}(\cdot|s_t)$ we sample the attacked action from the linear combination of both distributions using a hyperparameter $\alpha \in [0, 1]$ resulting in

$$a_t \sim (1 - \alpha)\pi^{(v)}(a_t|s_t) + \alpha\pi^{(a)}(a_t|s_t). \tag{27}$$

This enables our RAP version Algorithm 6 to attack the target agent using action adversarial attacks during data collection. In addition, during each episode in the environment we uniformly sample a different adversary uniformly from a population of n adversaries. The adversary population covers more potential vulnerabilities, thus forcing the target agent to generalize by punishing overfitting to a single adversary.

While the version by Vinitsky et al. [7] trains each adversary only on samples collected while that adversary was used, we choose to train each adversary on all samples by recomputing the probabilities for the chosen actions. The increased sample size allows for better learning and therefore stronger adversaries, requiring more robustness from the target agent.

5.4 Adversary Agnostic Policy Distillation

While the solutions so far employed adversaries during training to increase robustness, Qu et al. [8] aim to create a policy robust against state adversarial attacks without using a specific adversary. For this they use PD which allows for the training of a new student network recreating the behaviour of a different teacher network.

To train the student network $\pi_\theta^{(S)}$ with parameters θ using samples generated by the teacher network $\pi^{(T)}$ we require a loss function enforcing the wanted behaviour. As a result, A2PD [8] uses a loss function L^{A2PD} combining prescription gap maximization L^{pgm} with a Jacobian regularization L^{jr}

$$L^{\text{A2PD}} = L^{\text{pgm}} + \beta L^{\text{jr}}. \tag{28}$$

Given a state s_t in which the teacher chose action $a_t^{(T)}$, the prescription gap maximization L^{pgm} is minimized when the student has a very high preference for action $a_t^{(T)}$ while all other actions are equally unlikely to be chosen. This can be expressed by

$$L^{\text{pgm}}(\theta) = -\pi_\theta^{(S)}(a_t^{(T)}|s_t)^\eta \cdot \left[- \sum_{a \in \mathcal{A} \setminus \{a_t^{(T)}\}} \left(\frac{\pi_\theta^{(S)}(a|s_t)}{1 - \pi_\theta^{(S)}(a_t^{(T)}|s_t)} \right) \log \left(\frac{\pi_\theta^{(S)}(a|s_t)}{1 - \pi_\theta^{(S)}(a_t^{(T)}|s_t)} \right) \right] \tag{29}$$

Algorithm 6 RAP [7] integrated into the PPO version given in Algorithm 1.

Require: hyperparameters N_{iters}, T, N_{epochs}, n, α and environment *env*

Initialize policy actor network $\pi_\theta^{(v)}$ and with parameters θ
for $j = 1, \dots, n$ **do** ▷ we use a population of n adversaries
  Initialize adversary $\pi_{\omega_j}^{(a)}$ and with parameters ω_j
end for
for $\text{epoch} = 1, \dots, N_{\text{epochs}}$ **do**
  $\theta_{\text{old}} \leftarrow \theta$
  $\omega_{j,\text{old}} \leftarrow \omega_j$ for all $j = 1, \dots, n$
  $\mathcal{D}^{(v)} \leftarrow \{\}$ ▷ Initialize episode buffer for target
  $\mathcal{D}^{(a)} \leftarrow \{\}$ ▷ Initialize episode buffer for adversary
  for $t = 1, \dots, T$ **do** ▷ Collect T time steps of data
    if *env* has terminated **then**
      Sample uniformly to get $j \sim \text{Uniform}(1, n)$ ▷ Select adversary from population
      reset *env* and obtain new s_t
    end if
    Sample $a_t \sim (1-\alpha)\pi_{\theta_{\text{old}}}^{(v)}(a_t|s_t) + \alpha\pi_{\omega_{j,\text{old}}}^{(a)}(a_t|s_t)$ ▷ Sample action from combined distribution
    execute a_t in *env* and obtain r_t, s_{t+1}
    $\mathcal{D}^{(v)} \leftarrow \mathcal{D} \cup \{s_t, a_t, r_t, \pi_{\theta_{\text{old}}}^{(v)}(a_t|s_t)\}$
    $\mathcal{D}^{(a)} \leftarrow \mathcal{D} \cup \{s_t, a_t, -r_t, \pi_{\theta_{\text{old}}}^{(a)}(a_t|s_t)\}$
  end for
  Compute advantage estimates $\hat{A}_1, \dots, \hat{A}_T$ using critic
  for $\text{iter} = 1, \dots N_{\text{iters}}$ **do** ▷ Train agent using collected data
    Sample mini-batch from $\mathcal{D}$
    Optimize L^{CLIP} w.r.t. θ using mini-batch and SGD
  end for
end for

which is minimized if both the likelihood $\pi_\theta^{(S)}(a_t^{(T)}|s_t)$ is maximized and the entropy over all remaining actions are maximized. $\eta \in (0, 1)$ acts as a parameter in Equation 7 balancing the likelihood maximization and the entropy maximization. For our experiments we chose $\eta = \frac{1}{3}$ as Qu et al. [8] determined this to ensure a good balance. Since an adversarial attack needs to change the action selected by the agent to be effective, which would be less probable for a large prescription gap, minimizing this loss should increase agent robustness.

This loss is combined with the Jacobian regularization L^{jr} given by

$$L^{\text{jr}}(\theta) = \left\| \frac{\partial L^{\text{pgm}}(\theta)}{\partial s_t} \right\|_F . \tag{30}$$

This L^{jr} is minimized if the partial derivative is approaching 0 for each dimension of the state space, making it more difficult for small perturbations to each state space dimension to significantly affect the probability distribution output by the student network.

Additionally, the network size of the student network may be smaller than the network size of the teacher. This may be used to make the resulting agent faster to compute and to prevent overfitting caused by excessive network size.

Algorithm 7 states the A2PD based on [8]. Each epoch begins with the collection of sample data, created by letting the teacher play out episodes in the environment. The collected data is then used to train the student for several iteration.

Algorithm 7 A2PD [8] algorithm.

Require: hyperparameters N_{iter}, T, N_{epochs}, teacher policy $\pi^{(T)}$ and environment *env*
  Initialize student network $\pi_\theta^{(S)}$ and with parameters θ
  for $\text{epoch} = 1, \ldots, N_{\text{epochs}}$ **do**
    $\mathcal{D} \leftarrow \{\}$ ▷ Initialize episode buffer
    for $t = 1, \ldots, T$ **do** ▷ Collect T time steps of data using teacher
      if *env* has terminated **then**
        reset *env* and obtain new s_t
      end if
      $a_t \leftarrow \arg\max_a \pi^{(T)}(a|s_t)$
      execute a_t in *env* and obtain r_t, s_{t+1}
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t\}$
    end for
    for $\text{iteration} = 1, \ldots N_{\text{iter}}$ **do** ▷ Train student using collected data
      Sample mini-batch from $\mathcal{D}$
      Optimize L^{A2PD} w.r.t. θ using mini-batch and SGD ▷ L^{A2PD} given in Equation 28
    end for
  end for

6 Results

To evaluate the performance of different defensive methods we compare them using differnt attack methods and strategies. Besides the algorithms ATLA, RTGA, RAP and A2PD we also use an undefended PPO agent as a baseline. As a measure of performance we will mainly evaluate episode return achieved by the agents as the reward function of the environment correlates a high reward with good TN performance.

Each agent we evaluate has been trained using comparable parameters. A list of hyperparameters used in our experiments is given in the Appendix A.

In our analysis we focus first on the behavior exhibited by each specific algorithm during training and when attacked in Subsection 6.1 before looking at strategically timed attacks in Section 6.2. After this we do broader comparison between agents for each attack in Subsection 6.3.

6.1 Agent Behavior

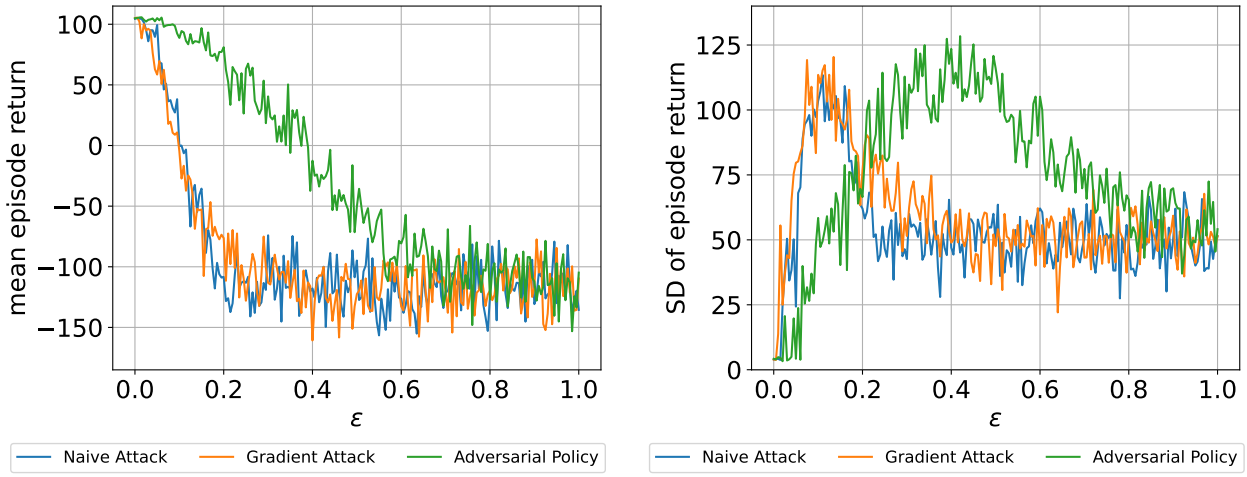
During training, RTGA and RAP required the most training time since RTGA needs to evaluate the policy multiple times to select the best noise in gradient direction and RAP requires a population of agents to be learned. ATLA also required more computation time than PPO but less compared to RAP or RTGA. A2PD was very quick to train but requires an already trained baseline PPO agent.

Due to high variance in episodes if the agent is attacked, each point in the following graphs uses a sample size of 100 episodes. Therefore, we focus on the mean and standard deviation of the respective return values to quantitatively evaluate the various defense mechanisms.

Figure 4 depicts all three attack methods evaluated using the PPO agent. It is evident that for the PPO both the naive and the gradient attack are highly effective. For $\epsilon = 0.2$ both methods have lowered the agents mean return to proximity of the floor imposed by the environment. The adversarial policy in comparison requires $\epsilon = 0.7$ to achieve minimum agent return. While very noisy, the curve of the mean return is almost linear for each attack until the return floor is reached and the mean remains constant.

There is also a strong correlation between the mean return curve and the standard deviation curve. The standard deviation is minimal for a very high agent return before increasing sharply as the agents performance decreases, reaching a maximum at about the same ϵ as is required for the mean reward to be 0. This is followed by a falling section before remaining constant once the standard deviation is about 50 and the mean is close to the floor. This may be explained by the nature of the environment. If the agent is capable of solving the environment it will find its way to the Target node very quickly. Otherwise it will wander around seemingly randomly until it stumbles into a state in which the way to the target node is clear to the agent. Due to this pattern, which remains valid throughout our experiments, other Figures will not depict standard deviation. Graphs of the standard deviation will be listed in the Appendix B instead.

Figure 5 depicts graphs for the mean return achieved by the various defensive agents, each figure containing the results for one agent evaluated using all uniform attack methods. The



(a) Mean episode reward for PPO during various uniform attacks.

(b) SD of episode reward for PPO during various uniform attacks.

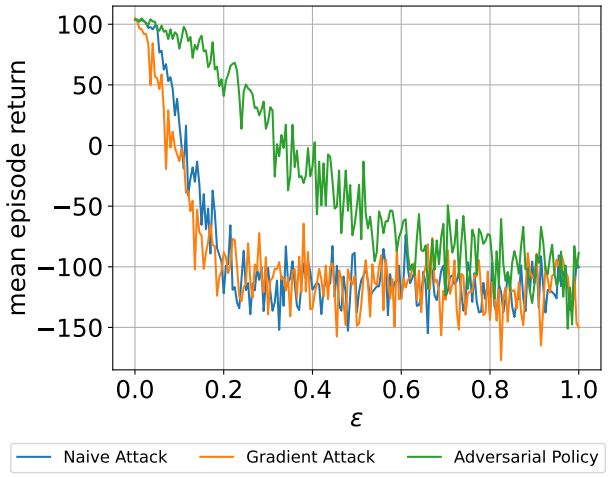
Figure 4. Episode returns plotted against varying ϵ using standard PPO. Both graphs depict different uniform attack methods. Left figure depicts mean while right figure depicts standard deviation.

behavior of the ATLA agent is very similar to that of the baseline PPO agent as indicated by Figure 5a. However, the other defensive methods show noticeable differences.

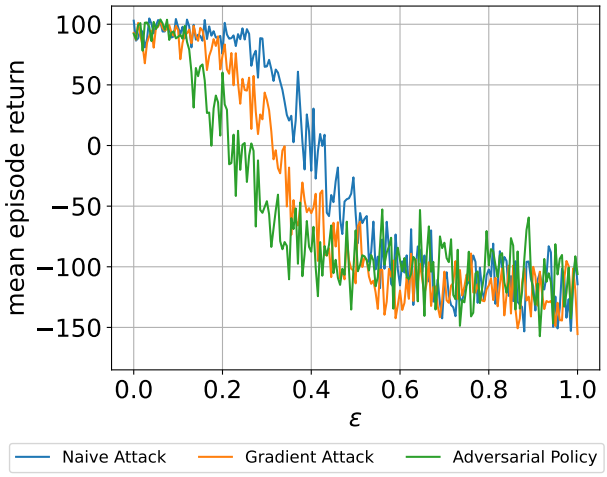
The RTGA agent in Figure 5b shows the same behavior for all attack methods, with a shift along the ϵ-axis indicating a difference in robustness. The adversarial policy is the most effective attack method against RTGA, even though it was the least effective attack method for both PPO and ATLA. The gradient attack is less effective but performs better than the naive attack. There is also a wide range for low values of ϵ during which the mean return of the attacked RTGA is almost equivalent to the return without attacks. This section then leads to a sharp decline in the mean return once ϵ exceeds this range. This stable range of ϵ is greater for the less effective attacks.

RAP in Figure 6 shows unexpected behaviors, achieving a comparatively lower mean return for $\epsilon = 0$. This is a major negative for the practical use of RAP as a not attacked agent would perform worse than other algorithms presented here. The adversarial policy attack follows a similar profile as for PPO or ATLA. The same holds true for the profile of the naive attack. Despite this, RAP behaves unexpectedly when subject to a gradient attack. Even though theory suggests the gradient attack to be more powerful than the naive attacks, applying the noise in gradient direction significantly reduces the impact of the attack, making the gradient attack achieve a lesser return reduction than the adversarial policy.

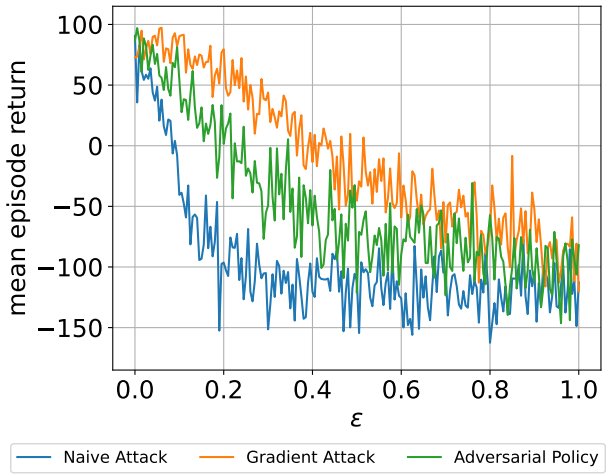
This difference between gradient and naive attacks does not exist for the A2PD agent, both attacks following the same pattern as seen in PPO or ATLA. As seen in Figure 5d, A2PD does, however, prove to be very robust against adversarial policy attacks. Even for $\epsilon = 1$ the adversarial policy attack does not manage to reduce the mean return below -50. This may be in part explained by the smaller network size reducing both overfitting as well as the potential for inherent weaknesses in the network.



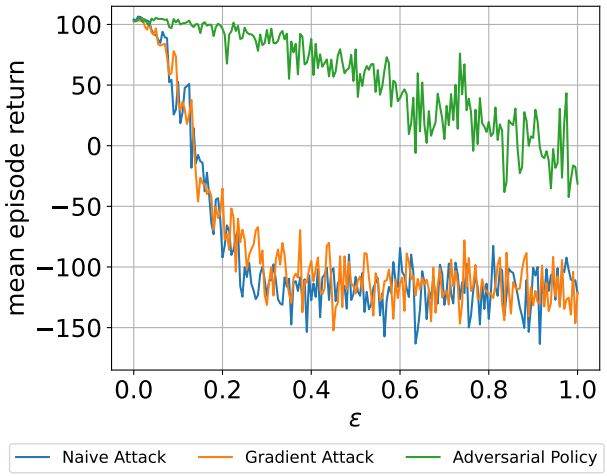
(a) Mean episode reward during uniform attacks for ATLA.



(b) Mean episode reward during uniform attacks for RTGA.



(c) Mean episode reward during uniform attacks for RAP.



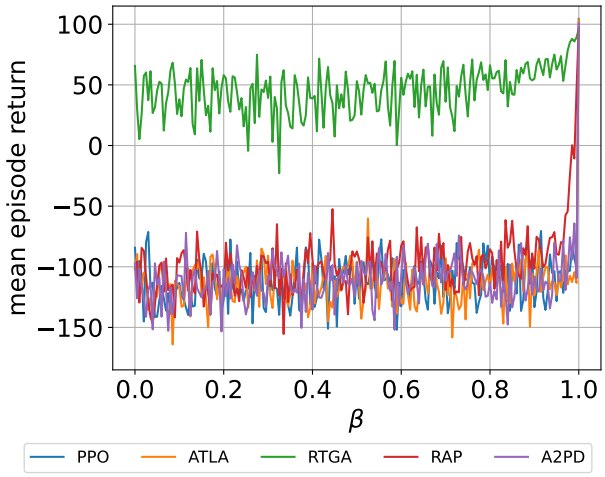
(d) Mean episode reward during uniform attacks for A2PD.

Figure 5. Mean episode returns plotted against varying ϵ. Each figure displays the results for a single algorithms during uniform attacks using all three attack methods.

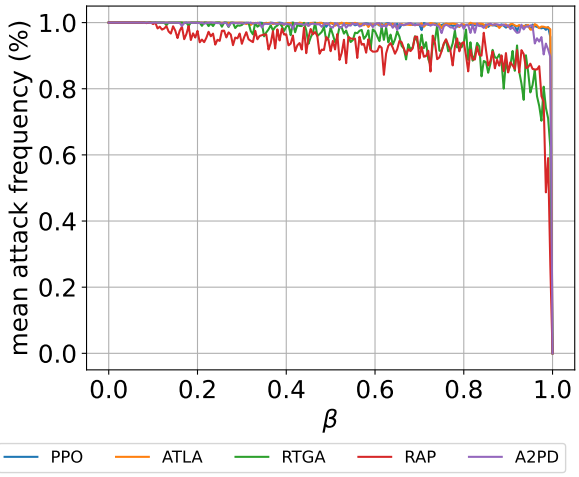
6.2 Differences of Attack Timings

During our experiments we examined the effects observed when using strategically timed attacks. These results are given in Figure 6 for varying values of β. Note that for $\beta = 0$ this attack timing becomes a uniform attack. For each attack two separate figures are given, one showing the mean return achieved by the agent and one showing the mean frequency of attacks. These figures include all defensive methods and the baseline PPO agent. Frequency of an episode is calculated as the number of time steps on which the strategically timed attack chose to attack during one episode divided by the total length of that episode. Each attack was applied with $\epsilon = 0.35$.

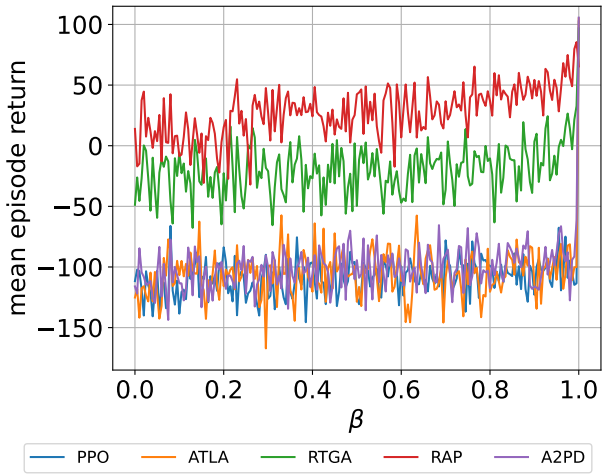
When examining the mean episode return for all attacks and algorithms in Figures 6a, 6c, 6e a pattern becomes apparent. For all combinations of attack method and defense the mean



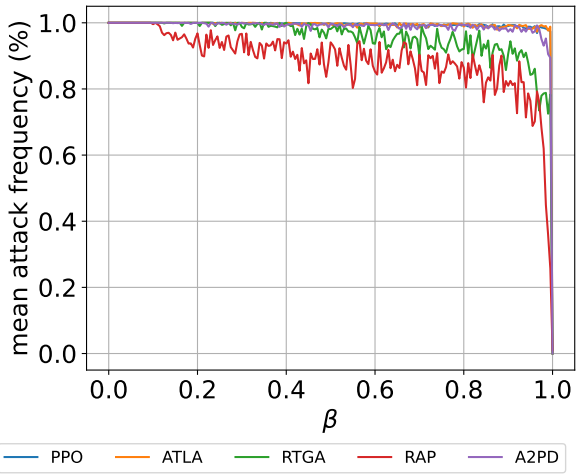
(a) Mean episode reward during strategically timed naive attack.



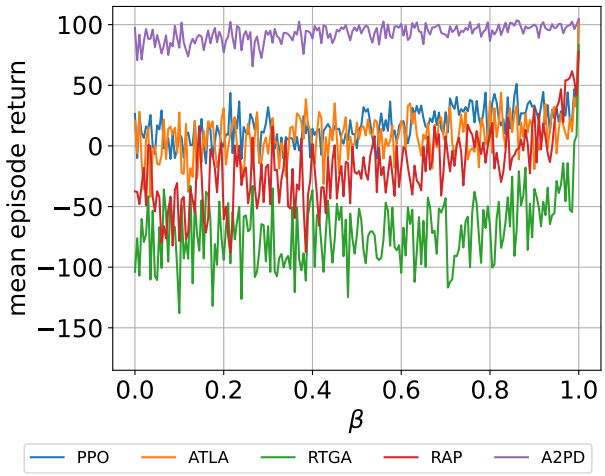
(b) Mean attack frequency during strategically timed naive attack.



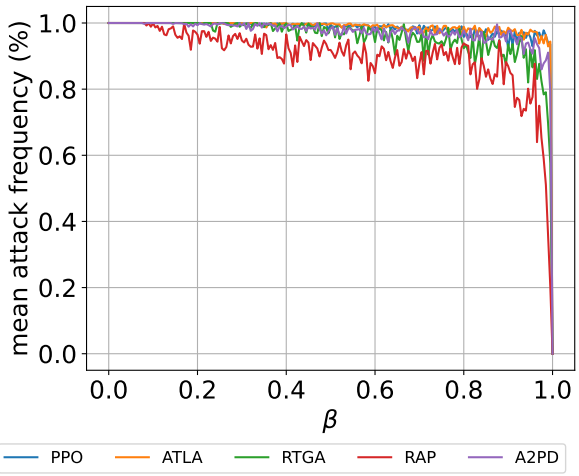
(c) Mean episode reward during strategically timed gradient attack.



(d) Mean attack frequency during strategically timed gradient attack.



(e) Mean episode reward during strategically timed adversarial policy attack.



(f) Mean attack frequency during strategically timed adversarial policy attack.

Figure 6. Mean episode returns and attack frequencies plotted against varying β for $\epsilon = 0.35$. Each figure displays the results for all algorithms during the same strategically timed attack. Attack frequency is given in percentage of time steps attacked per episode.

return only marginally increases until β approaches 1 which causes to the strategic timing to never choose to attack, making the mean return jump up to the values achieved for the same algorithm without any attack.

Similar patterns exist for frequency. All combinations exhibit a very high attack frequency with only a sharp drop of for β values close to 1. Despite this there is some variation. RTGA and RAP appear to be attacked less frequently than the other algorithms. In particular during the naive attack RAP is attacked even less frequently than RTGA. This indicates that these two algorithms do not show a strong preference for the best action, since otherwise the attack frequency would be similar to the other defenses.

Another difference is seen between the frequency of the noise based attacks (naive and gradient) compared to the frequency of the adversarial policy attacks. While the noise based attacks do not lower the preference function for PPO, ATLA and A2PD, the adversarial policy attack does decrease the preference. This is indicated by the difference in the frequency graphs for the mentioned algorithms between noise based and adversarial policy attacks. During the adversarial policy attacks the frequency decreases slightly even for comparatively lower β values.

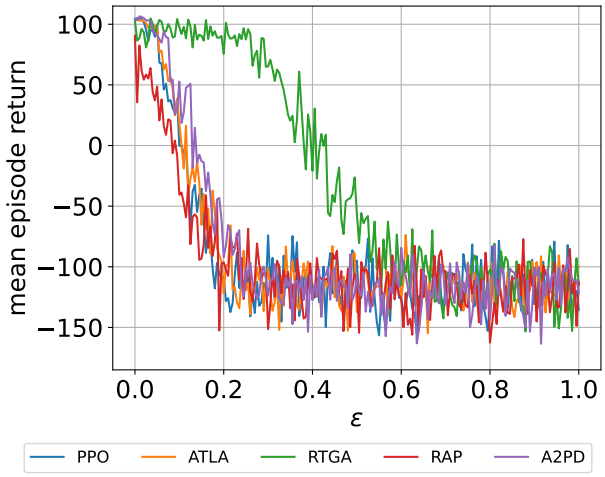
While these differences exist, they appear to only be minor. Therefore the rest of this investigation will focus on uniform attacks. This allows us to interpret the results of other experiments more easily due to the reduced complexity. More work is needed to determine if other attack timings would show a greater difference with regards to defenses or attack methods.

6.3 Agent Comparison

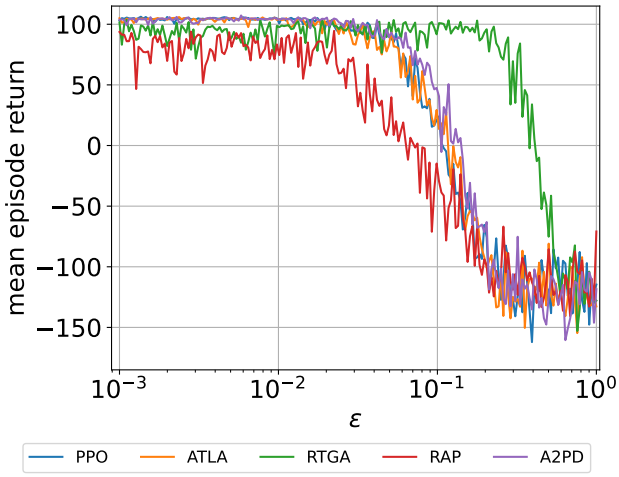
We now know the strengths and weaknesses of each solution viewed in isolation and have evaluated the differences in attack timings. This allows us to compare the algorithms by looking at each specific uniform attack method. The empirical results are visualized in Figure 7. Each attack is shown in two versions, one with a linear and one with a logarithmic scale for ϵ since the logarithmic scale is more helpful to differentiate the algorithms for lower values of ϵ.

First, we will look at the performance of the metrics during a uniform naive attack in Figure 7a and Figure 7b. On the linear scale it is obvious how much more robust RTGA is against naive attacks when compared to the other algorithms, achieving a stable performance even for $\epsilon = 0.25$. The other algorithms perform significantly worse even for lower ϵ. In particular, the logarithmic plot indicates RAP as the least robust for low ϵ and A2PD is only a minimal increase in robustness compared to both PPO and ATLA.

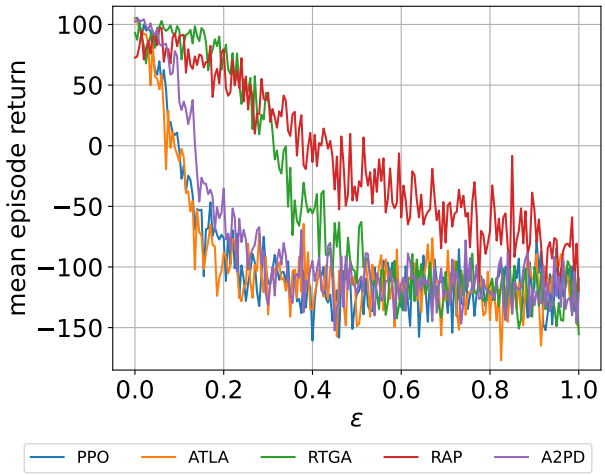
Observing the gradient attack in Figure 7c and Figure 7d, we find RAP has lower default performance for very low ϵ to contrast with its unrivaled robustness for $\epsilon > 0.35$. While suffering greater mean return reductions for $\epsilon > 0.35$, RTGA proves the best algorithm for lower ϵ and remains better than ATLA, PPO and A2PD for $\epsilon < 0.5$ even after being surpassed by RAP. PPO and ATLA seem to perform comparable to each other just as during the naive attack, both suffering the greatest return reduction for the gradient attack. A2PD is noticably better than PPO and ATLA but does not manage to match the robustness of RTGA or RAP.



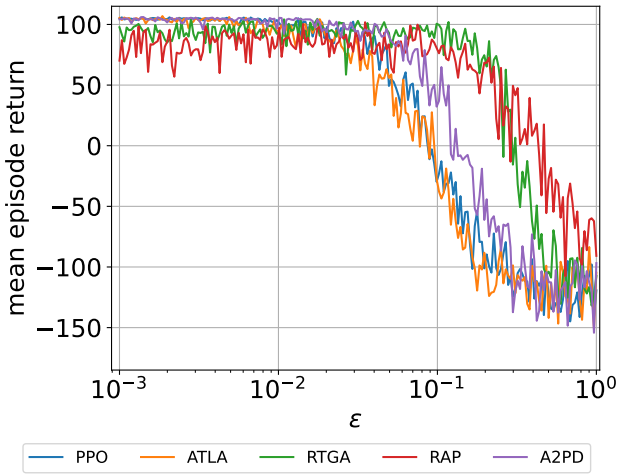
(a) Mean episode reward during uniform naive attack on linear scale.



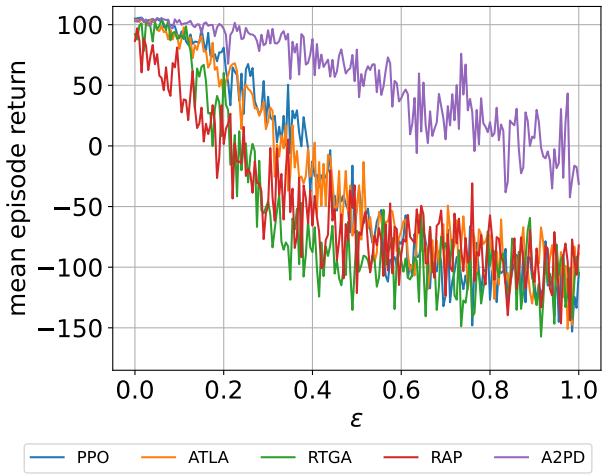
(b) Mean episode reward during uniform naive attack on logarithmic scale.



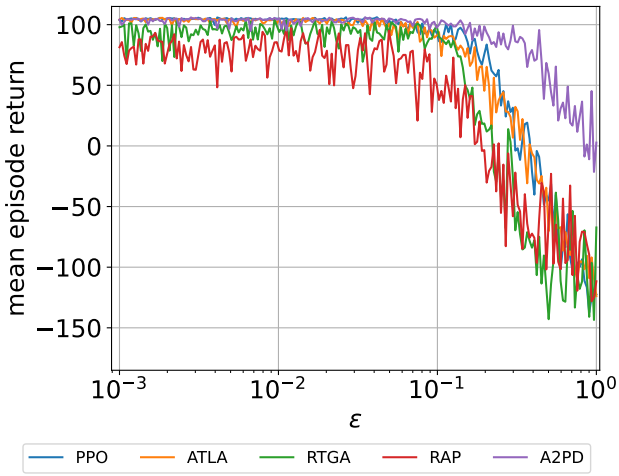
(c) Mean episode reward during uniform gradient attack on linear scale.



(d) Mean episode reward during uniform gradient attack on logarithmic scale.



(e) Mean episode reward during uniform adversarial policy attack on linear scale.



(f) Mean episode reward during uniform adversarial policy attack on logarithmic scale.

Figure 7. Mean episode returns plotted against varying ϵ. Each figure displays the results for all algorithms during the same uniform attack.

Lastly, we evaluate the agents during the adversarial policy attack in Figure 7e and Figure 7f. A2PD is clearly the best algorithm here, remaining stable for lower ϵ and being the most robust by a wide margin for large ϵ. Once again PPO and ATLA achieve similar mean returns, yet both outperform RTGA and RAP. For $\epsilon < 0.2$ RTGA manages to be more robust than RAP but for greater ϵ they both perform equally poorly.

The comparisons in Figure 7 show robustness of a defensive algorithm can not be quantified by only observing the qualities it exhibits under a single attack method. The different algorithms offer different advantages and disadvantages.

We found in our experiments ATLA to be the least notable defensive method. For all attack methods it matched the performance of the baseline PPO agent, neither being superior nor inferior. Hence, ATLA fails to increase robustness of the agent in our TN environment.

RTGA provides strong robustness against noise based adversarial attacks but fails to retain performance during adversarial policy attacks. This might hint at the existence of intrinsic vulnerabilities in the network trained through this algorithm. While the applied gradient attack during training leads to robustness against noise based attacks, the weakness to adversarial policies might be more important since adversarial policies are much quicker to evaluate, making such attacks more realistic for practical use by a malicious attacker.

Those concerns regarding adversarial policy attacks are also present for RAP. Further, RAP also exhibits a strong vulnerability to naive attacks. This behaviour is counter-intuitive since the gradient attack was devised as an improvement upon the naive attack The strong robustness when faced with noise based gradient attacks suggests some resistance to noise based attacks, yet this is seemingly disproven by the lack of robustness against naive attacks which are purely injecting noise. One explanation could be that the agent only learns to resist perturbations applied in the negative gradient direction while remaining vulnerable to perturbations in the opposite direction.

A2PD is the least effected algorithm with regards to intrinsic weaknesses of the network architecture. The smaller network size as well as the specific training of the student network lead to an agent which is highly robust against adversarial policies. However, when subjected to noise based attacks A2PD is less resistant, even though it still manages to achieve higher returns compared to the baseline PPO agent. Future investigations might look into adding noise to the samples used for the student's training, aiming thereby to achieve similar robustness against noise based attacks as is present in RTGA.

We can thus answer Problem 4.1. Except for RAP all defensive methods managed to retain performance for $\epsilon = 0$ comparable to that of a not attacked PPO agent in the same TN environment. In addition, A2PD increased agent robustness for all attacks compared to the baseline PPO. Thus A2PD is one solution for Problem 4.1. RTGA is also very promising as it achieves excellent robustness against both noise based attacks, only suffering from a vulnerability to adversarial policy attacks. ATLA is not sufficient since our experiments showed no increase in robustness compared to the baseline and RAP is only robust against gradient attacks, performing worse than the baseline on both adversarial policies and naive attacks.

As an answer to Problem 4.2, A2PD is likely more suitable for TN development as it does not make the agent more vulnerable against any attack method. Additionally, it is also quick to train and facilitates smaller neural network sizes for the underlying architecture. This

should also speed up the evaluation of the policy when the system is deployed in the field, further increasing network speed. A2PD's ease of implementation is a further benefit since it can be used to defend any RL agent. A2PD is not perfect as can be seen in the much greater noise robustness of RTGA. An ideal algorithm would seek to integrate the noise resistance of RTGA into A2PD.

7 Conclusion

This thesis has examined various defensive methods for robust RL in the context of routing in TNs. We have introduced key concepts for understanding RL and adversarial attacks. To model the dynamic challenges of TNs we have also presented an RL environment reflecting these circumstances, which was the result of our previous work [3].

Next, we stated the research question this thesis seeks to answer as two formal problems in Section 4 before detailing the four defensive solutions intended as solutions. The investigated solutions covered a wide range of approaches. RAP [7] and RTGA [6] served as representative of adversarial training in our experiments. Similarly we utilized ATLA [5] as an example for a robust learning scheme and A2PD [8] as a variation of defensive distillation. For this thesis we used each defense to protect a PPO [13] agent operating in the previously established TN environment. This required adjustments to the various algorithms which were detailed in 5.

During our experiments we empirically tested the performance of the various defenses while subject to different attack methods and attack timings. We found strategically timed attacks to not be very successful in reducing the frequency of attacks required to reduce the victim agents performance. The most significant effects were achieved for RAP and RTGA where the strategic timing lead to slight reduction in attack frequency. Future work would might need to employ more complex timing strategies to regulate attack frequency as strategically timed attacks proved insufficient against the agents in our TN environment.

By varying the strength of the attacks under a uniform timing through adjusting ϵ we also gained insights into the strengths and weaknesses of each specific algorithm. A2PD proved to be the best overall solution by improving robustness for all attacks but for noise based attack methods the slight improvements resulting from A2PD were less meaning full compared to the large robustness increase provided by RTGA. In turn, RTGA was shown to decrease agent robustness for adversarial policy attacks, a vulnerability not exhibited by A2PD. Since adversarial policy attacks are the most likely attack to be put into practise, this is of major importance. Training time required also varied between algorithms, A2PD proving to be very quick to train in contrast to the expensive training of RTGA and RAP.

In our previous work [3] we developed an RL environment for routing in TNs, yet the trained RL agent proved susceptible to adversarial attacks and failed to meet the reliability and security requirements for practical usage. This thesis addressed this issue by seeking more robust RL algorithms. To mitigate the vulnerability to adversarial attacks, we have identified a defensive solution capable of increasing agent robustness across all tested adversarial attack methods. Further we have assessed the advantages and disadvantages of other defense methods, paving the way for future research seeking to improve upon the best existing defense A2PD. This marks this thesis as a vital step towards safe and secure RL routing in TNs. Future work can use the results presented here as a basis for the development of secure agents featuring even greater robustness.

References

[1] J. F. Loevenich, R. R. F. Lopes, P. H. Rettore, S. M. Eswarappa, and P. Sevenich, "Maximizing the probability of message delivery over ever-changing communication scenarios in tactical networks," *IEEE Networking Letter*, vol. 3, no. 2, pp. 84–88, 2021.

[2] J. F. Loevenich, P. H. Rettore, R. R. F. Lopes, and A. Sergeev, "A bayesian inference model for dynamic neighbor discovery in tactical networks," *Procedia Computer Science*, vol. 205, pp. 28–38, 2022, 2022 International Conference on Military Communication and Information Systems (ICMCIS).

[3] J. F. Loevenich, J. Bode, T. Hürten, L. Liberto, F. Spelter, P. H. Rettore, and R. R. F. Lopes, "Adversarial attacks against reinforcement learning based tactical networks: A case study," in *MILCOM 2022-2022 IEEE Military Communications Conference (MILCOM)*. IEEE, 2022, pp. 986–992.

[4] S. H. Huang, N. Papernot, I. J. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *CoRR*, vol. abs/1702.02284, 2017. [Online]. Available: http://arxiv.org/abs/1702.02284

[5] H. Zhang, H. Chen, D. S. Boning, and C. Hsieh, "Robust reinforcement learning on state observations with learned optimal adversary," *CoRR*, vol. abs/2101.08452, 2021. [Online]. Available: https://arxiv.org/abs/2101.08452

[6] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust deep reinforcement learning with adversarial attacks," *CoRR*, vol. abs/1712.03632, 2017. [Online]. Available: http://arxiv.org/abs/1712.03632

[7] E. Vinitsky, Y. Du, K. Parvate, K. Jang, P. Abbeel, and A. M. Bayen, "Robust reinforcement learning using adversarial populations," *CoRR*, vol. abs/2008.01825, 2020. [Online]. Available: https://arxiv.org/abs/2008.01825

[8] X. Qu, Y. Ong, A. Gupta, and Z. Sun, "Defending adversarial attacks without adversarial attacks in deep reinforcement learning," *CoRR*, vol. abs/2008.06199, 2020. [Online]. Available: https://arxiv.org/abs/2008.06199

[9] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6199

[10] I. Ilahi, M. Usama, J. Qadir, M. U. Janjua, A. I. Al-Fuqaha, D. T. Hoang, and D. Niyato, "Challenges and countermeasures for adversarial attacks on deep reinforcement learning," *CoRR*, vol. abs/2001.09684, 2020. [Online]. Available: https://arxiv.org/abs/2001.09684

[11] H. Zhang, H. Chen, C. Xiao, B. Li, D. S. Boning, and C. Hsieh, "Robust deep reinforcement learning against adversarial perturbations on observations," *CoRR*, vol. abs/2003.08938, 2020. [Online]. Available: https://arxiv.org/abs/2003.08938

[12] S. Kaviani, B. Ryu, E. Ahmed, K. A. Larson, A. Le, A. Yahja, and J. H. Kim, "Robust and scalable routing with multi-agent deep reinforcement learning for manets," *CoRR*, vol. abs/2101.03273, 2021. [Online]. Available: https://arxiv.org/abs/2101.03273

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[15] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1506.02438

[16] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[17] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[19] Y. Lin, Z. Hong, Y. Liao, M. Shih, M. Liu, and M. Sun, "Tactics of adversarial attack on deep reinforcement learning agents," *CoRR*, vol. abs/1703.06748, 2017. [Online]. Available: http://arxiv.org/abs/1703.06748

[20] J. F. Loevenich, A. Sergeev, P. H. L. Rettore, and R. R. F. Lopes, "An intelligent model to quantify the robustness of tactical systems to unplanned link disconnections," in *2022 18th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2022, pp. 1–8.

A Hyperparameters & Network Architecture

The following lists the hyperparameters used in our experiments:

- Discount factor $\gamma = 0.99$.
- For all gradient and naive attacks (Algorithm 2, 3):
  - $n = 10$.
  - $\alpha = \beta = 0.5$.
- PPO [13] (Algorithm 1):
  - $T = 4000$.
  - $N_{\text{iters}} = 80$.
  - $N_{\text{epochs}} = 1000$.
- RTGA (Algorithm 4):
  - $T = 4000$.
  - $N_{\text{iters}} = 80$.
  - $N_{\text{epochs}} = 1000$.
  - $\epsilon = 0.2$ for the gradient attack.
- ATLA (Algorithm 5):
  - $T = 4000$.
  - $N_{\text{epochs}} = 1000$.
  - $N_{\text{iters}}^{(v)} = 80$.
  - $N_{\text{iters}}^{(a)} = 80$.
- RAP (Algorithm 6):
  - $T = 4000$.
  - $N_{\text{iters}} = 80$.
  - $N_{\text{epochs}} = 1000$.
  - $n = 3$.
  - $\alpha = 0.2$.
- A2PD (Algorithm 7):
  - $T = 40000$.
  - $N_{\text{iters}} = 800$.
  - $N_{\text{epochs}} = 100$.

  - $\eta = \frac{1}{3}$ in Equation 29.
  - The trained PPO agent was used as the teacher.

PPO, ATLA, RTGA, RAP and all adversarial policies used the same network architecture. After each layer a $\tanh$ was used as the activation function. Excluding the input and output layers, the network architecture in order was given by:

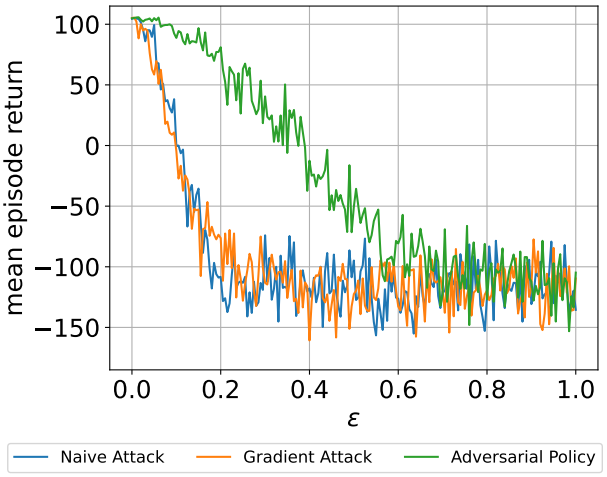
- Fully connected layer with 128 neurons.
- Fully connected layer with 128 neurons.
- Fully connected layer with 64 neurons.

The student network for A2PD used a reduced architecture consisting of:

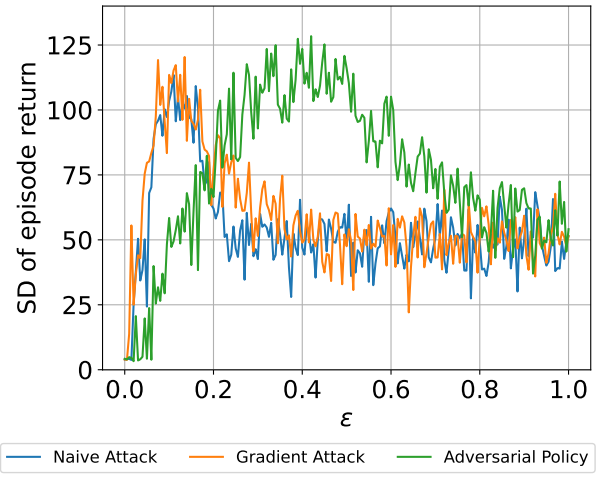
- Fully connected layer with 128 neurons.
- Fully connected layer with 64 neurons.

B Experiment Graphs

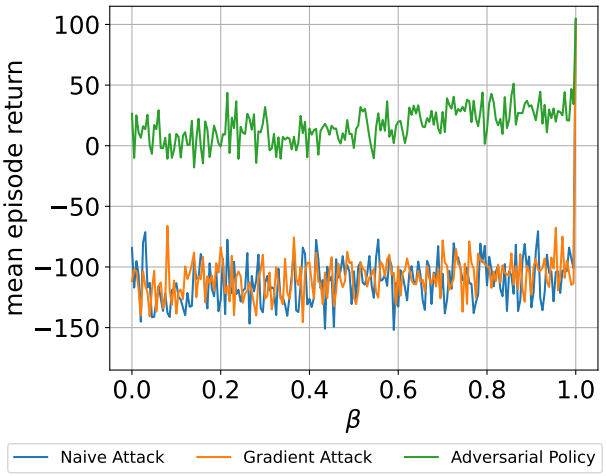
The following will show various graphs resulting from our experiments using uniform and strategically timed attacks.



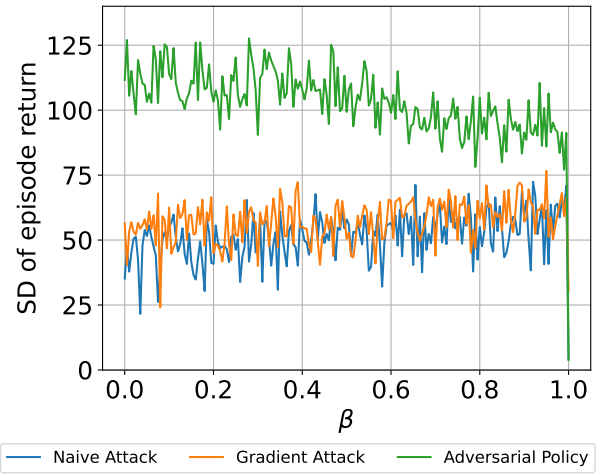
(a) Mean episode return for all uniform attacks.



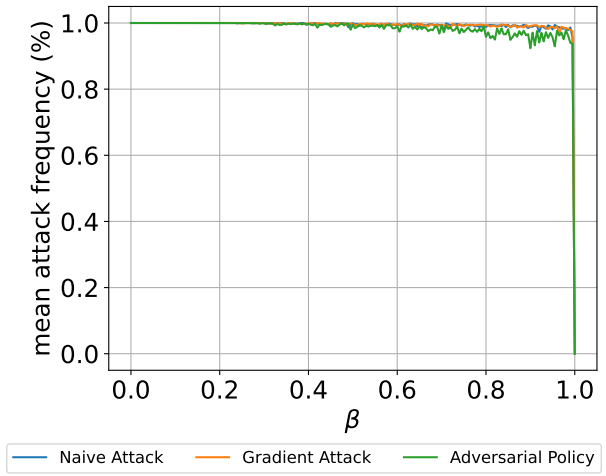
(b) Standard deviation of the episode return for all uniform attacks.



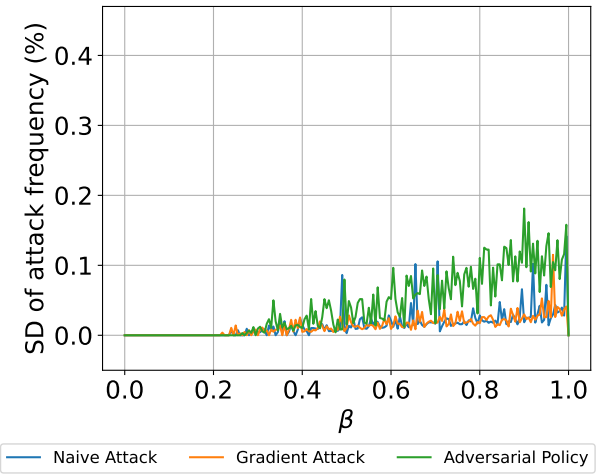
(c) Mean episode return for all strategically timed attacks.



(d) Standard deviation of the episode return for all strategically timed attacks.

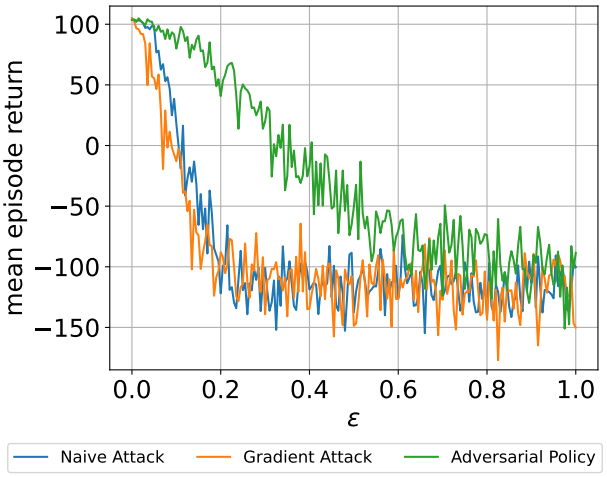


(e) Mean attack frequency for all strategically timed attacks.

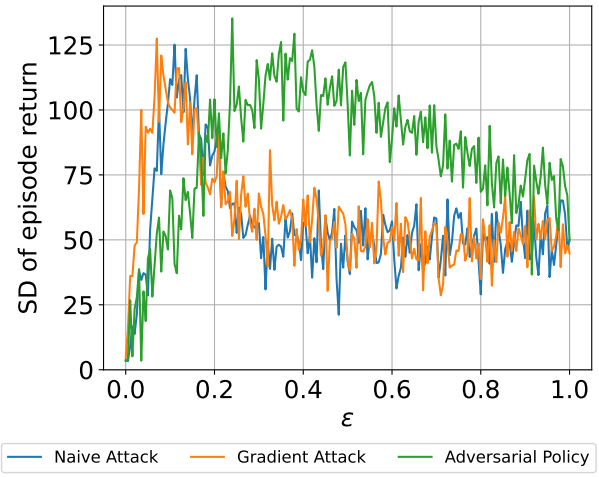


(f) Standard deviation of the attack frequency for all strategically timed attacks.

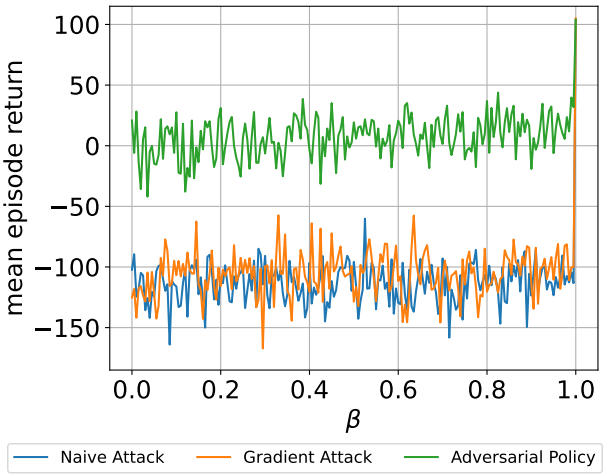
Figure 8. Experiments for PPO



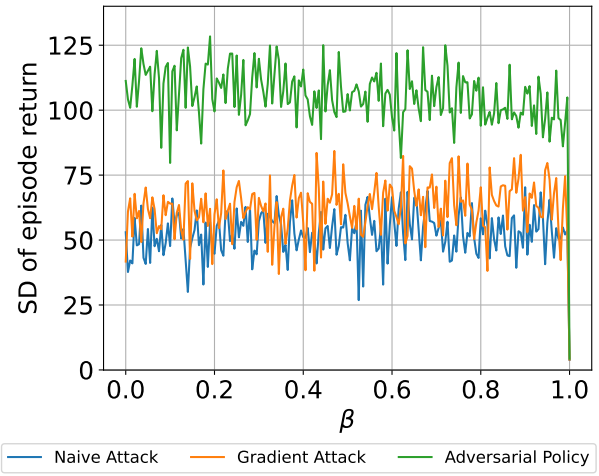
(a) Mean episode return for all uniform attacks.



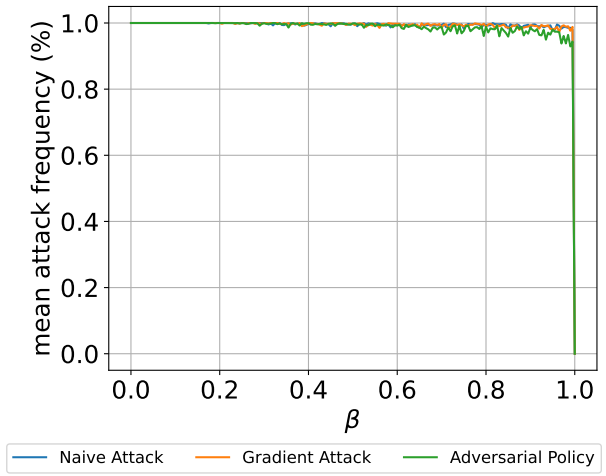
(b) Standard deviation of the episode return for all uniform attacks.



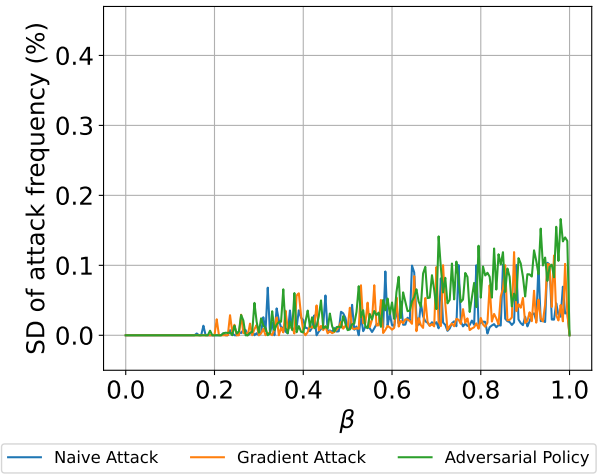
(c) Mean episode return for all strategically timed attacks.



(d) Standard deviation of the episode return for all strategically timed attacks.

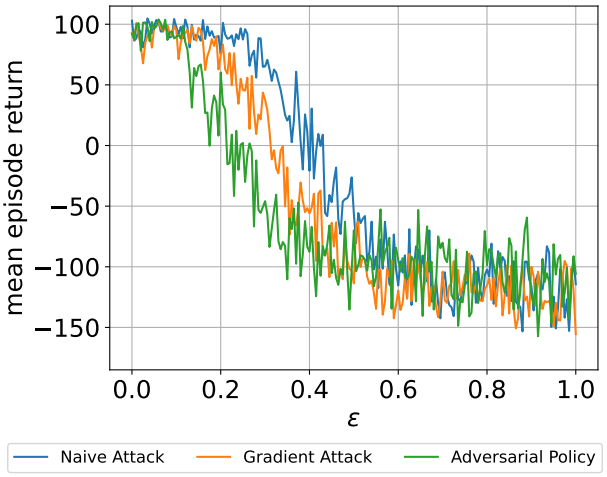


(e) Mean attack frequency for all strategically timed attacks.

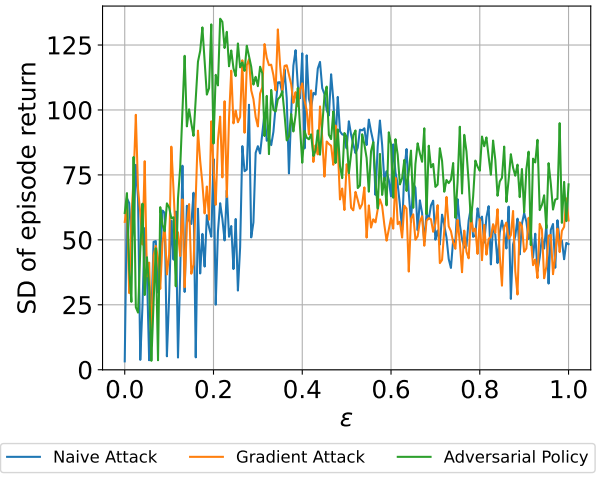


(f) Standard deviation of the attack frequency for all strategically timed attacks.

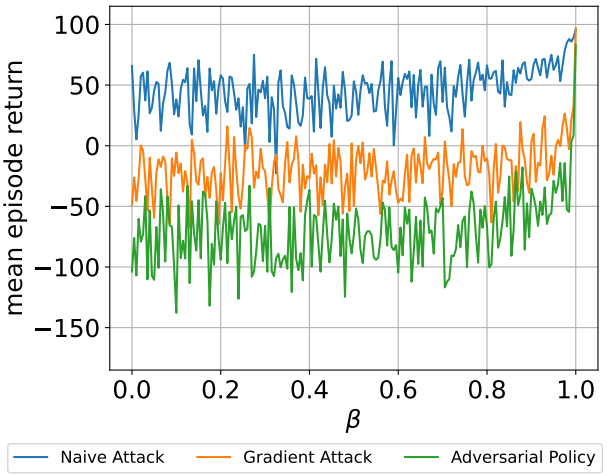
Figure 9. Experiments for ATLA



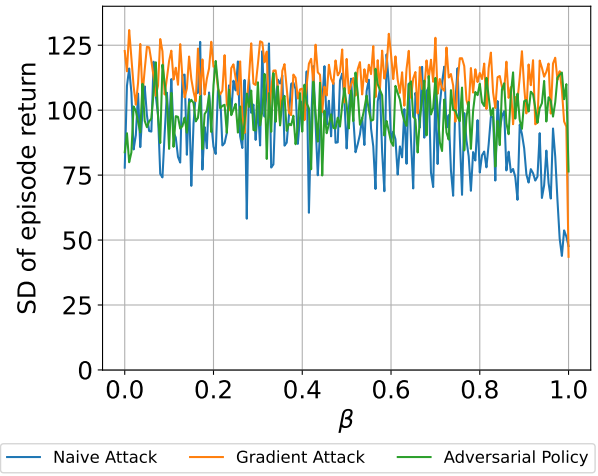
(a) Mean episode return for all uniform attacks.



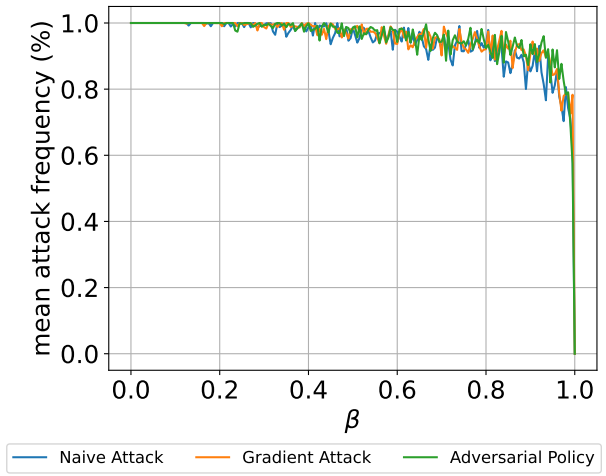
(b) Standard deviation of the episode return for all uniform attacks.



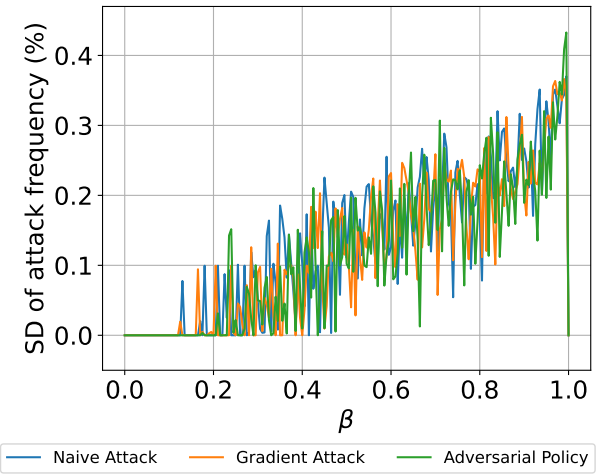
(c) Mean episode return for all strategically timed attacks.



(d) Standard deviation of the episode return for all strategically timed attacks.

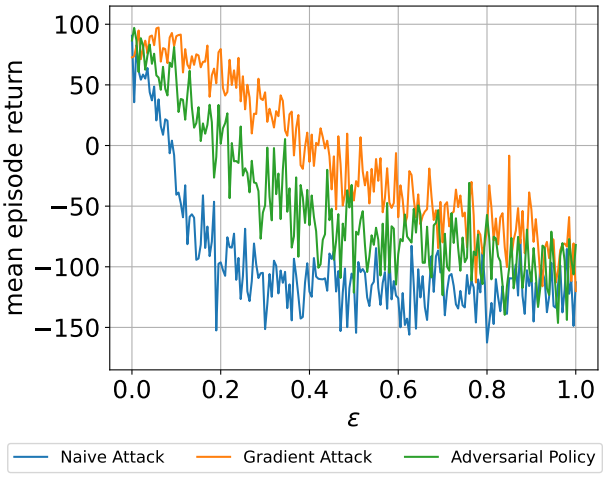


(e) Mean attack frequency for all strategically timed attacks.

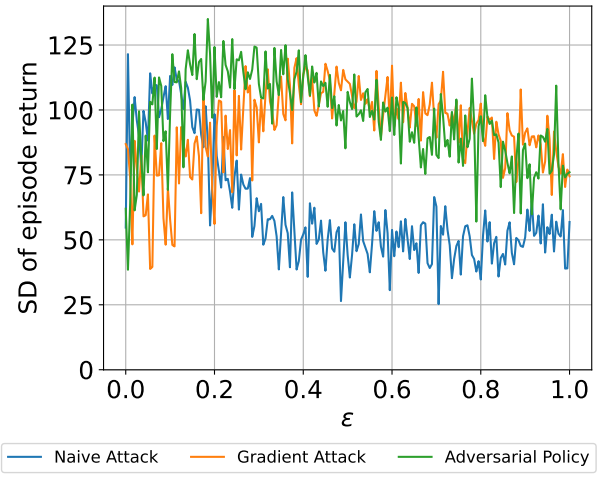


(f) Standard deviation of the attack frequency for all strategically timed attacks.

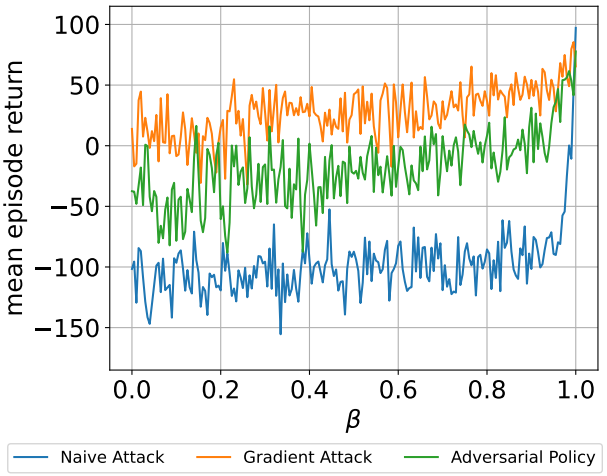
Figure 10. Experiments for RTGA



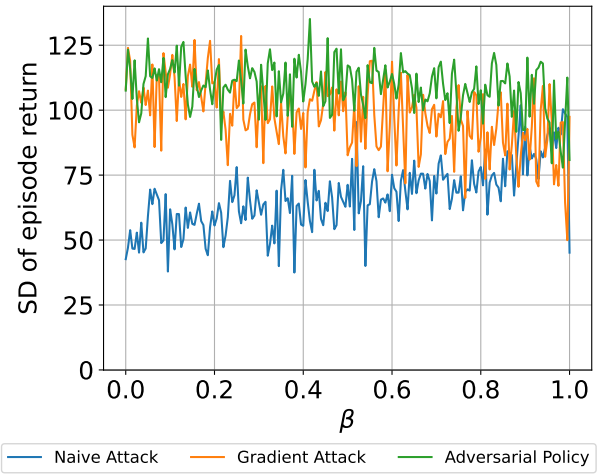
(a) Mean episode return for all uniform attacks.



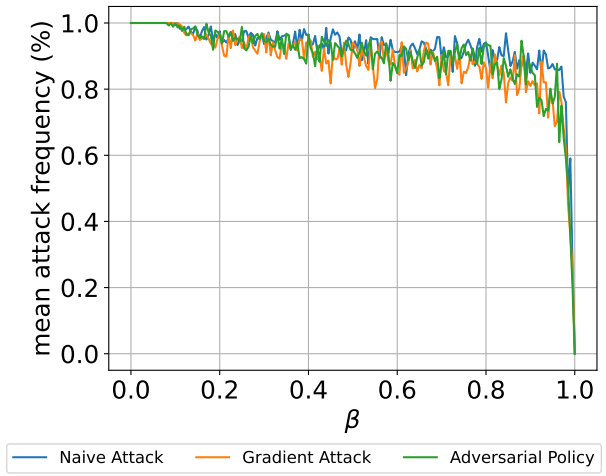
(b) Standard deviation of the episode return for all uniform attacks.



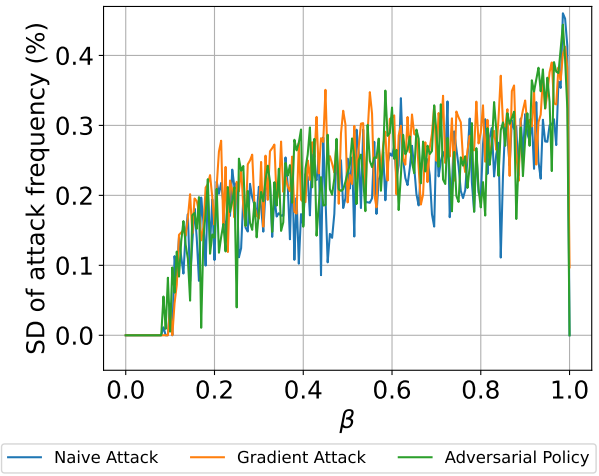
(c) Mean episode return for all strategically timed attacks.



(d) Standard deviation of the episode return for all strategically timed attacks.

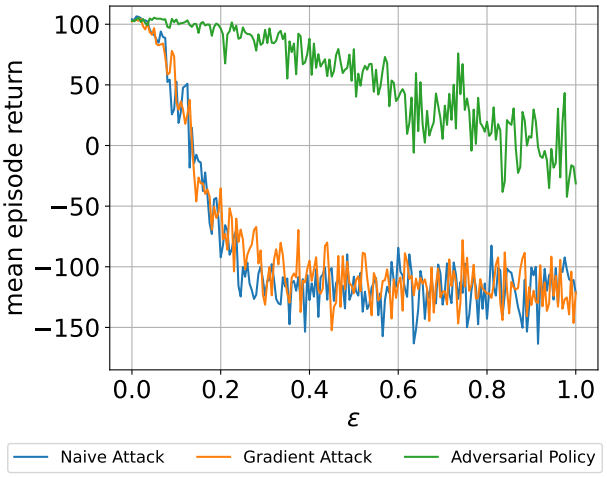


(e) Mean attack frequency for all strategically timed attacks.

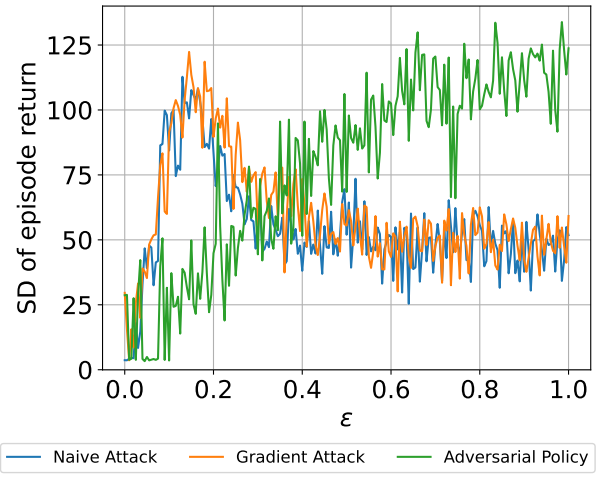


(f) Standard deviation of the attack frequency for all strategically timed attacks.

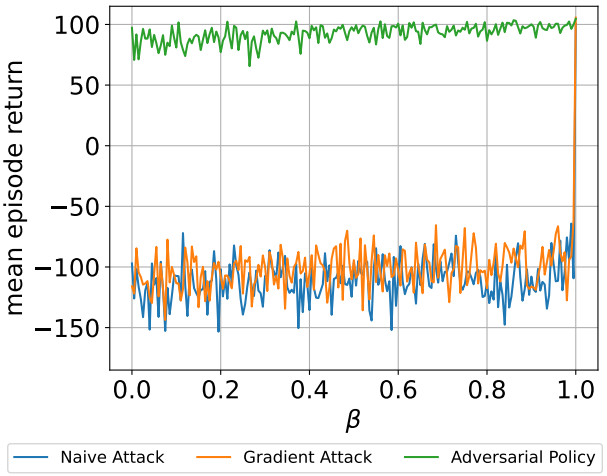
Figure 11. Experiments for RAP



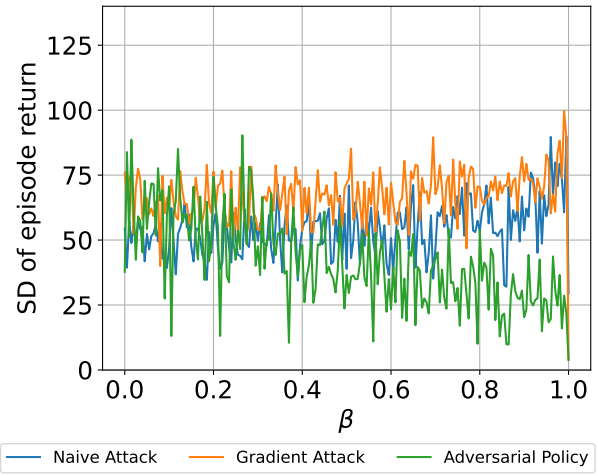
(a) Mean episode return for all uniform attacks.



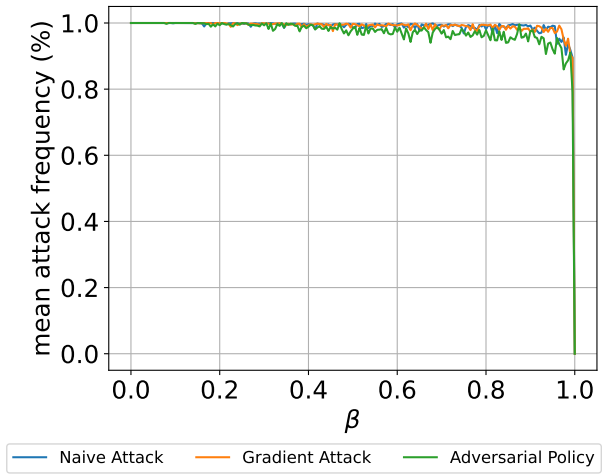
(b) Standard deviation of the episode return for all uniform attacks.



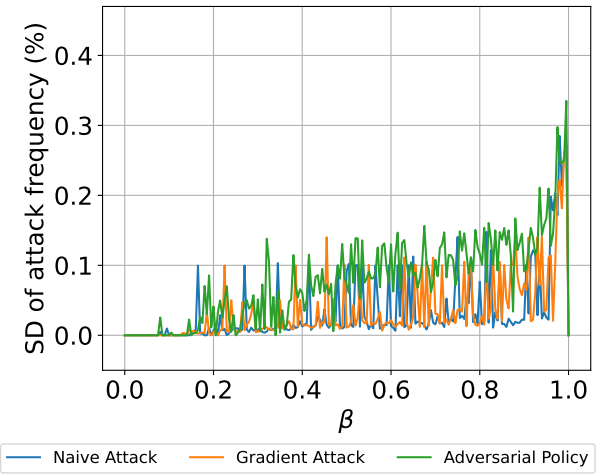
(c) Mean episode return for all strategically timed attacks.



(d) Standard deviation of the episode return for all strategically timed attacks.

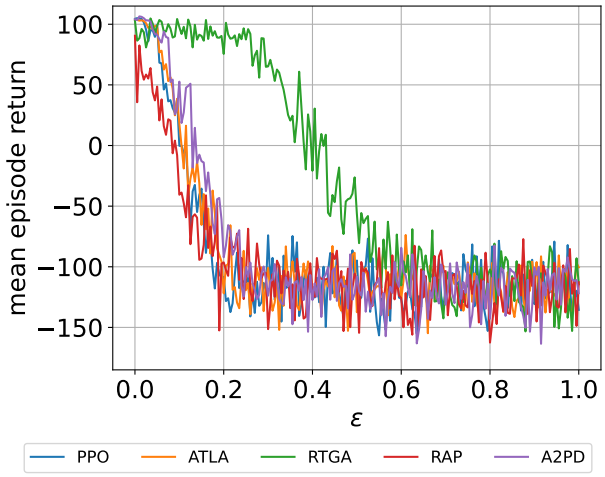


(e) Mean attack frequency for all strategically timed attacks.

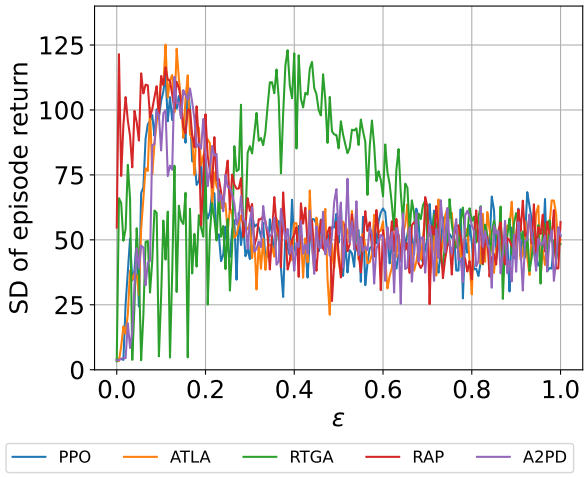


(f) Standard deviation of the attack frequency for all strategically timed attacks.

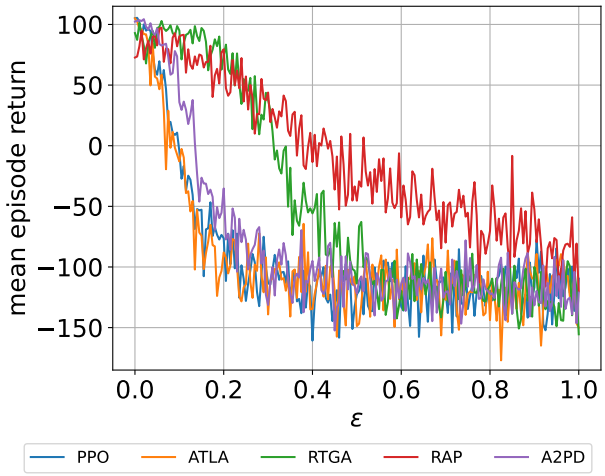
Figure 12. Experiments for A2PD



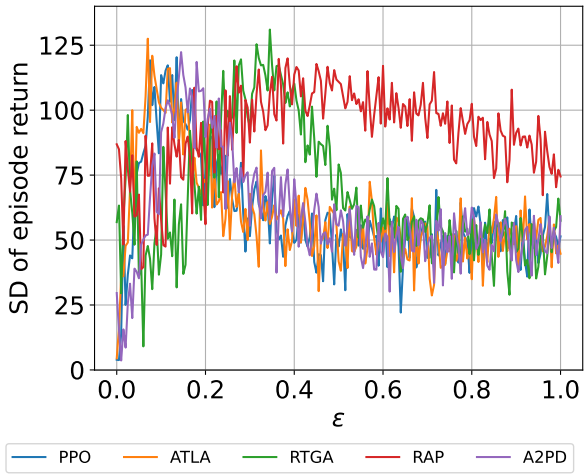
(a) Mean episode return for the naive attack.



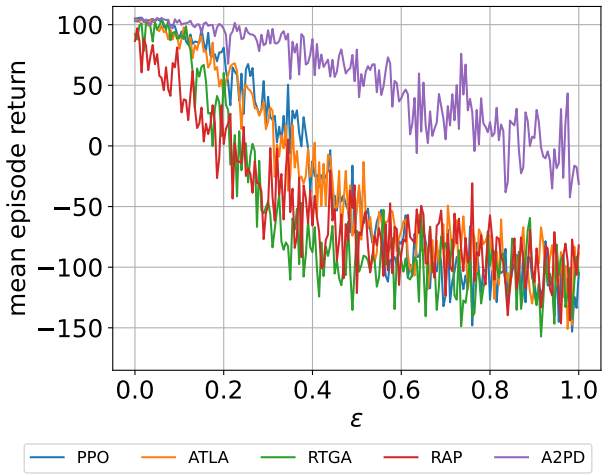
(b) Standard deviation of the episode return for the naive attack.



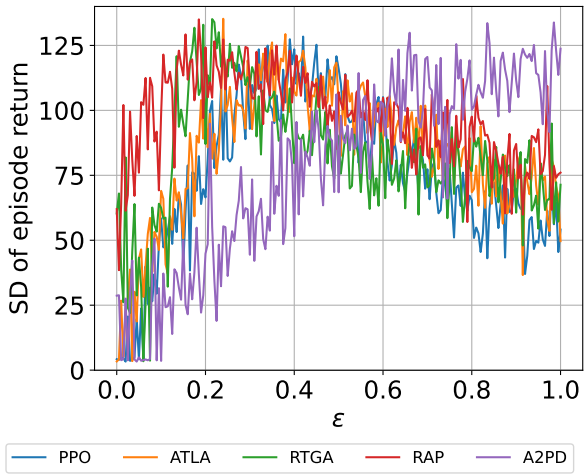
(c) Mean episode return for the gradient attack.



(d) Standard deviation of the episode return for the gradient attack.

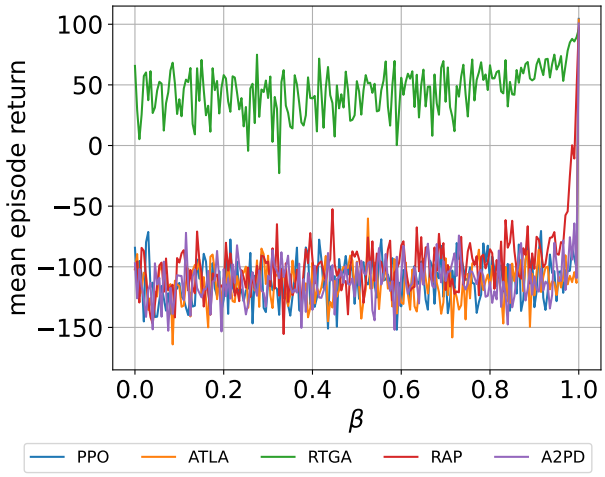


(e) Mean episode return for the adversarial policy attack.

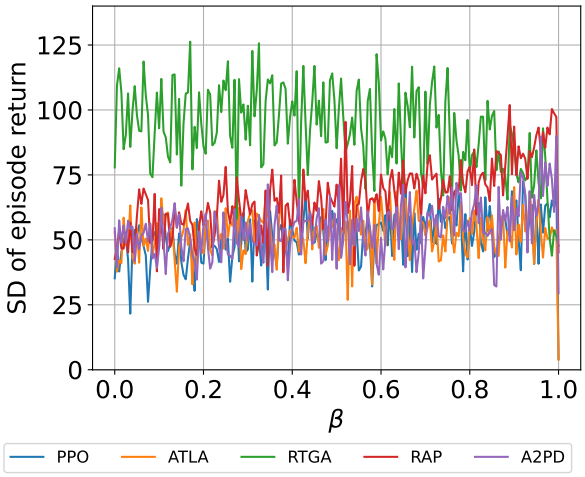


(f) Standard deviation of the episode return for the adversarial policy attack.

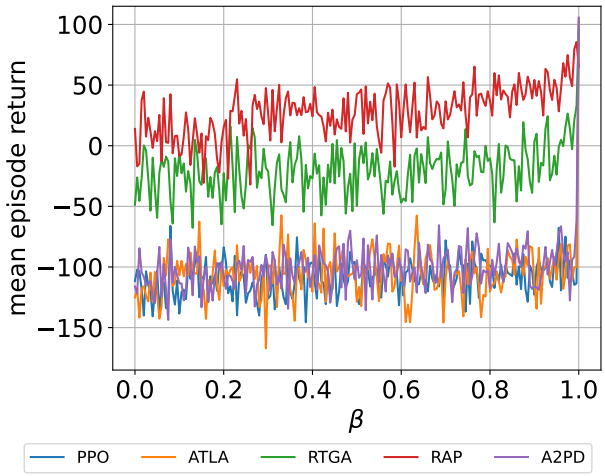
Figure 13. Episode return for each uniform attack.



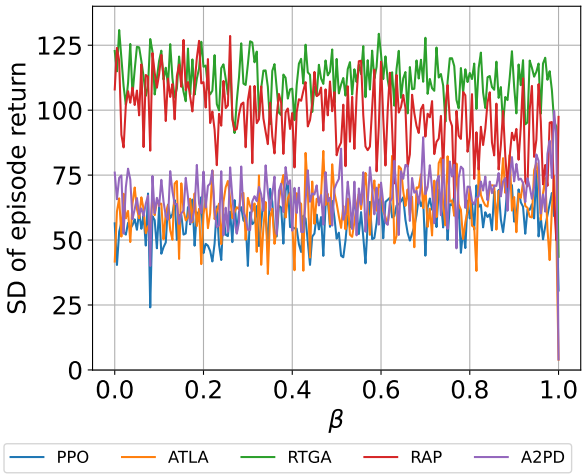
(a) Mean episode return for the naive attack.



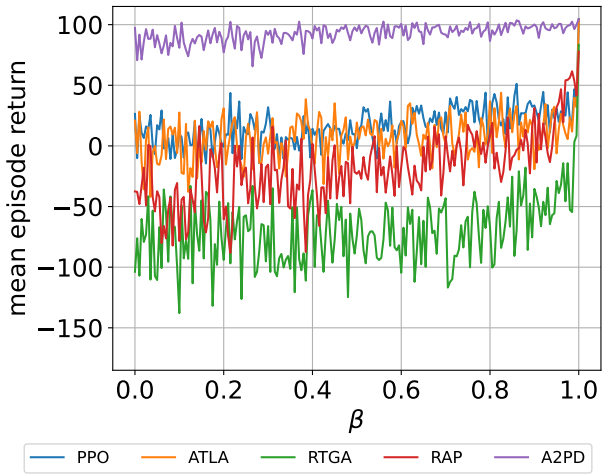
(b) Standard deviation of the episode return for the naive attack.



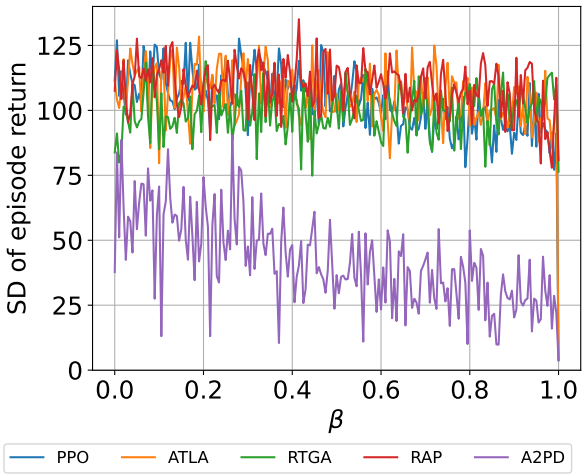
(c) Mean episode return for the gradient attack.



(d) Standard deviation of the episode return for the gradient attack.

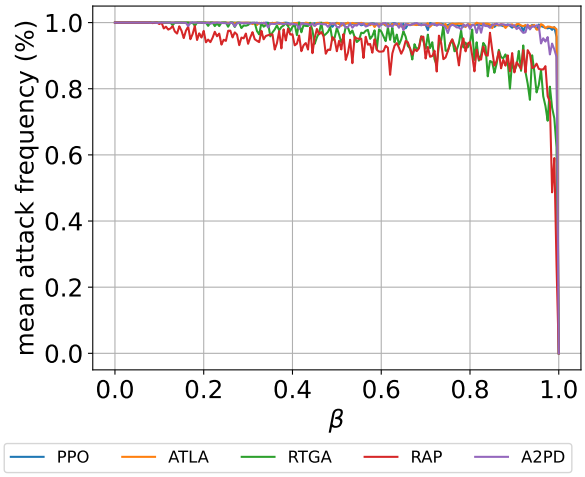


(e) Mean episode return for the adversarial policy attack.

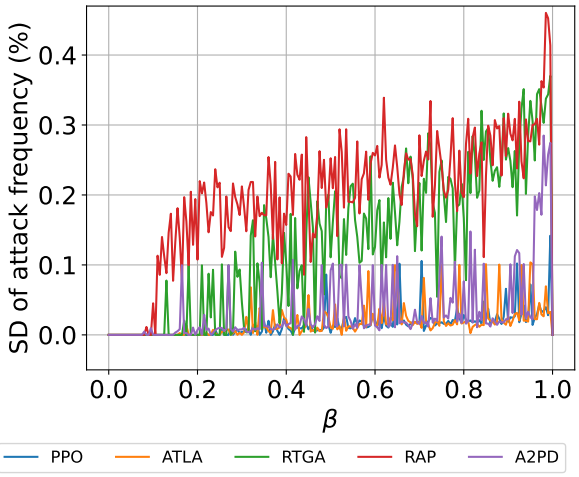


(f) Standard deviation of the episode return for the adversarial policy attack.

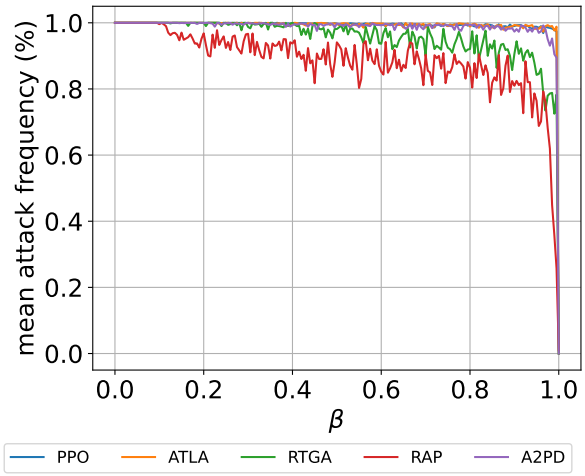
Figure 14. Episode return for each strategically timed attack.



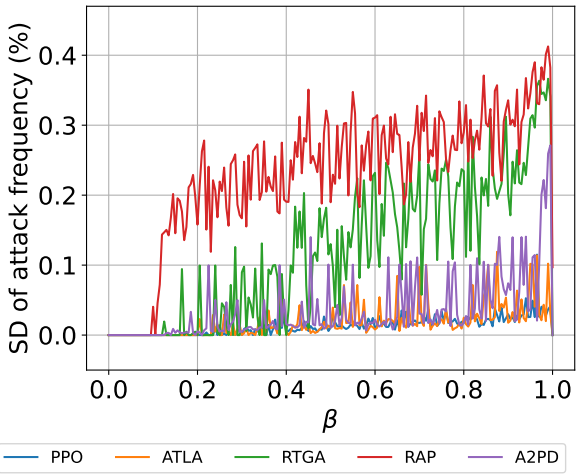
(a) Mean attack frequency for the naive attack.



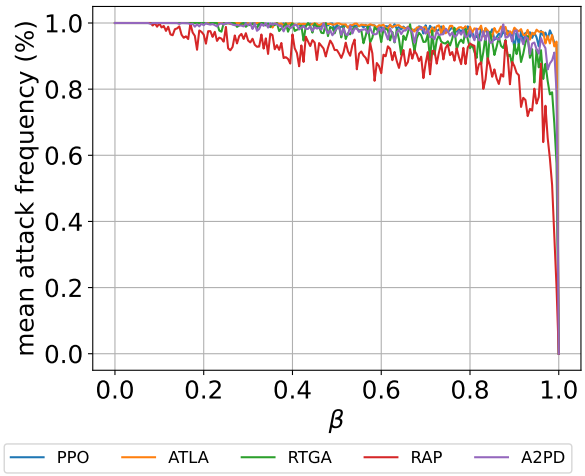
(b) Standard deviation of the attack frequency for the naive attack.



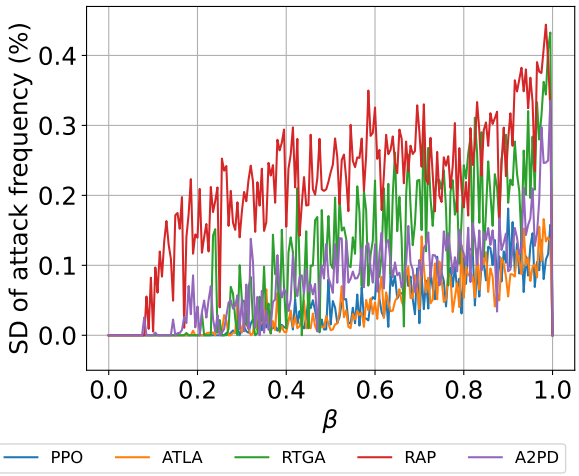
(c) Mean attack frequency for the gradient attack.



(d) Standard deviation of the attack frequency for the gradient attack.



(e) Mean attack frequency for the adversarial policy attack.



(f) Standard deviation of the attack frequency for the adversarial policy attack.

Figure 15. Attack frequency for each strategically timed attack.