AALBORG UNIVERSITET
STUDENTERRAPPORT

Computer Science
Aalborg University
https://www.aau.dk

TRIDENT: Integrating Predictive Residual Detection into Recurrent PPO for Adversarial Robustness

By Emil Gade, Frederik Brandi Nielsen, Kasper Vestergaard Jensen, Martin P. Tielemans, Magnus Storgaard, Raman Dara Aziz

Supervised by Yushuai Li

Abstract

In real-world and safety-critical environments, adversarial perturbations to sensor observations can cause Reinforcement Learning (RL) agents to behave unpredictably. To address this, we propose TRIDENT, a triple-headed reinforcement learning architecture that couples a recurrent Proximal Policy Optimization (PPO) agent with an integrated observation-space perturbation detector. A shared Long Short-Term Memory (LSTM) backbone feeds three heads: policy, value, and a predictive detector that classifies if the current observation is perturbed or not using a calculated residual. In training, Fast Gradient Sign Method (FGSM) attacks are applied to observations in the Halfcheetah-v5 environment, and in inference the detection accuracy is evaluated. Benchmarks show strong robustness under attack: our trained agent maintains high and stable rewards with detection accuracy near 98%, whilst heavily outperforming a baseline PPO that degrades and exhibits high variance. In clean conditions the baseline PPO exhibits higher peak returns, revealing a trade-off where our method prioritizes stability and robustness. Ablations indicate that recurrence is vital for learning features that support both control and reliable detection, while the detection head successfully identifies attacks without substantially degrading performance.

Disclaimer: Generative AI tools were used to assist with grammatical correction and text formulation in this paper. Furthermore it was also used for assistance during code development.

1 Introduction

Deep Reinforcement Learning (DRL) has seen major progress in recent years. Notably, Deep Q-Networks (DQNs) enabled agents to learn Atari games directly from raw pixels [1]. Since then, DRL methods have been successfully applied to continuous-control benchmarks based on the MuJoCo physics simulator, where algorithms such as Proximal Policy Optimization (PPO) achieve stable on-policy training [2]. As a result, DRL is now widely used for robotics, autonomous systems and other real world control tasks. However, transferring policies from simulation to the real world exposes them to sensor noise, model mismatch, and uncertainty. In safety-critical settings, even small state deviations can lead to unsafe behavior, making it crucial to both detect and withstand these variances [3].

Studies show that DRL agents are vulnerable to deliberately crafted adversarial perturbations. Small, norm-bounded input changes can drastically alter network outputs [4]. Furthermore, in RL, adversarial interference during interaction can severely degrade performance or cause total failure [3]. Several robust RL methods address this by training against an explicit adversary or optimizing for worst-case dynamics [3], [5]. While these approaches improve policy resilience, they generally do not explicitly detect attacks, treating robustness as a policy property rather than a distinct capability that can identify when perturbations occur. Alternatively, anomaly detection methods in industrial systems use recurrent models to predict observations, flagging large residuals as potential attacks [6]. While these exploit temporal structure to identify perturbations, they are typically trained separately from the policy, complicating their integration with online continuous control.

Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM), excel at modeling temporal dependencies. Memory-based control has been shown to improve performance in partially observable and noisy environments [7]. This suggests that recurrent architectures can be leveraged not only for control but also for detecting adversarial perturbations that disrupt normal temporal patterns.

Therefore the work on this paper focuses on two problems:

- Developing a predictive head that can detect when state perturbations occur (primary focus).
- Leveraging a shared recurrent backbone so that policy, value estimation, and detection are learned jointly, with the aim of achieving robustness to adversarial state perturbations.

We investigate whether integrating adversarial detection directly into a recurrent DRL agent improves robustness in continuous control. The proposed algorithm, TRIDENT, uses a shared backbone recurrent PPO architecture where a single LSTM layer feeds actor, critic, and predictive detection heads. The detection head predicts the next observation and analyzes the residual to identify adversarial attacks. The agent is trained jointly under FGSM-based attacks in HalfCheetah-v5 using a multi-objective loss that combines PPO, value estimation, and detection objectives.

2 Related Works

Given our primary goal, similar approaches have been widely explored in prior research on robust RL and adversarial state perturbations, making it relevant to explore related works in order to show how this project differentiates itself from existing research.

2.1 Adversarial Attacks

Qiaoben et al. [8] provides an analysis of adversarial attacks on state observations in DRL. They categorize various attack types, such as targeted and untargeted perturbations, and evaluate their effects across multiple RL agents.

Their findings show that even small attacks in the observation can degrade policy performance significantly. Their study primarily characterizes attack dynamics rather than proposing defensive strategies. While these findings are fundamental for this project, our work extends the idea of detection of adversarial attacks within a model-free PPO framework.

2.2 Robust-oriented Approaches

Zhang et al. [9], investigates how to make RL agents more robust to perturbations or attacks on their observations. Their approach introduces a regularization term during training that stabilizes the policy under slightly perturbed observations, thereby making the agent more robust. This work is particularly relevant to ours as their method is applied to the PPO algorithm in a continuous action space, similar to the HalfCheetah-v5 environment used in this project. Unlike their approach, our method does not rely solely on regularizing the policy but instead incorporates an explicit detection mechanism that identifies when perturbations occur.

Furthermore, Liang et al. [5] proposes a worst-case-aware training framework that achieves robustness without explicitly generating adversarial samples. Their method optimizes the policy under estimated worst-case conditions, effectively improving sample efficiency and generalization to unseen perturbations. In contrast, our approach incorporates a predictive detection head into PPO, allowing the agent to identify adversarial deviations online rather than relying purely on worst-case policy resilience.

2.3 Detection Based Approaches

The approach proposed by Zizzo et al. [6] resembles the predictive residual based detection method explored in this paper, as it also incorporates a LSTM based prediction framework for detecting adversarial perturbations. Their model predicts the next sensor reading from previous observations and uses the prediction residuals as an anomaly score. Large residuals show that the observation no longer follows the expected time-dependent pattern of the system, indicating there might be an attack. The authors evaluate their detector under gradient-based perturbations such as Fast Gradient Sign Method (FGSM), identical to the adversarial attack used in this paper. While their framework is designed for anomaly detection in static monitoring systems rather than RL, the underlying methods and idea aligns closely with what this paper proposes. Our model integrates a predictive detection head into a recurrent PPO with LSTM backbone that will predict the next observation, compute a residual, and analyze it using a dedicated neural classifier to estimate attack probability. Unlike their work, which relies on residual magnitude thresholding for offline anomaly detection, our method learns a residual-based decision boundary jointly with the policy and value networks, thus allowing for an online adversarial detection, and achieving robustness during interaction with the environment.

While these related works demonstrate improvements in robustness to adversarial inputs and detecting them, our project differs by integrating a dedicated detection module directly within the RL architecture. This detector leverages temporal information from a recurrent backbone to compare predicted and observed states over time, allowing the agent to detect perturbed states during interaction with the environment.

3 Preliminaries

3.1 Reinforcement Learning

Reinforcement learning formalizes sequential decision-making as an interaction between an agent and an environment, modeled as a Markov Decision Process (MDP). An MDP is defined by the tuple (S, A, p, R, γ), where S is the state space, A is the action space, $p(s', r \mid s, a)$ is the transition probability function, R

denotes the reward function, and $\gamma \in [0, 1]$ is the discount factor [10]. The agent's objective is to learn a policy π_θ that maximizes the expected discounted return, expressed as:

$$V_\theta(s) = \mathbb{E}_{\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

3.2 Proximal Policy Optimization

PPO is an on-policy RL algorithm that optimizes the policy through iterative updates while constraining policy changes to maintain training stability. PPO uses a clipped surrogate objective that is formulated as a loss function to be minimized [2]:

$$\mathcal{L}^{CLIP}(\theta) = -\,\mathbb{E}_t\big[\min\big(r_t(\theta)\hat{A}_t, \\ \text{clip}(r_t(\theta), 1-\epsilon_{clip}, 1+\epsilon_{clip})\hat{A}_t\big)\big]$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ is the probability ratio and $\hat{A}_t$ is the advantage estimate. The policy parameters θ are updated by minimizing $\mathcal{L}^{CLIP}(\theta)$ via gradient descent. Also, by utilizing the clipping mechanism, large destabilizing policy updates are prevented [2].

3.3 Generalized Advantage Estimation

Generalized Advantage Estimation (GAE) addresses the bias–variance tradeoff by combining information from k-step returns. It does so using a temporal-difference approach parameterized by the discount factor γ and the trace-decay parameter $\lambda_{GAE} \in [0, 1]$ [11]. The advantage estimate is expressed as a discounted sum of temporal-difference residuals δ^V_{t+l}, which measure the error between the predicted value and the 1-step bootstrap target:

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda_{GAE})} = \sum_{l=0}^{\infty} (\gamma\lambda_{GAE})^l \delta^V_{t+l}$$

Setting $\lambda_{GAE} = 0$ yields low variance estimates, while setting $\lambda_{GAE} = 1$ yields low bias estimates. Therefore, an intermediate value of λ_{GAE} balances the bias-variance trade-off, achieving optimal performance by combining the benefits of both extremes [11].

3.4 Adversarial Attacks and Robustness in RL

A critical aspect of RL agents is their vulnerability when transferred from simulation to the real world, where policies may overfit to the training environment and fail to generalize [3]. Adversarial attacks expose this weakness by perturbing the observed state s_t to produce a manipulated version $\tilde{s}_t = s_t + \delta_t$, where δ_t is a deliberately crafted perturbation. Robustness can be achieved by training agents under such adversarial condition or incorporation detection and defense mechanism, that enables the policy to remain stable under bounded perturbation [9].

3.5 Fast Gradient Sign Method

Fast Gradient Sign Method is a single-step adversarial attack designed to generate adversarial examples that degrade agent performance in neural networks. The attack operates by calculating the gradient of the model's loss function with respect to the input observation. It then perturbs the input by a small magnitude ϵ_{FGSM} in the direction of the sign of this gradient, which maximizes the loss and drives the model toward a suboptimal control action. The mathematical formulation for generating the adversarial observation $\tilde{s}$ from a clean input s and a target signal y is as follows:

$$\tilde{s} = s + \epsilon_{FGSM} \cdot \text{sign}\left(\nabla_s J(\theta, s, y)\right)$$

where $J(\theta, x, y)$ is the loss function, θ represents the model parameters, and sign() operator specifies the direction in which the loss increases [4].

3.6 Long Short-Term Memory

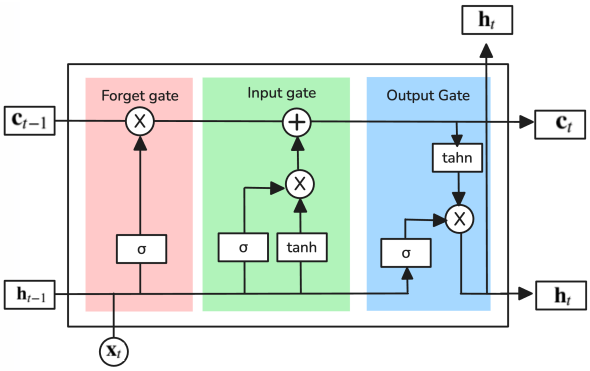

Figure 1: LSTM cell architecture. Adapted from Figure 2 in [12].

LSTM is a type of Recurrent Neural Network (RNN) designed to retain information over longer sequences. Standard RNNs struggle with long-term dependencies due to the vanishing gradient problem, where gradients shrink during backpropagation and prevent meaningful weight updates [13].

As shown in Figure 1, LSTMs address this by maintaining two states: the cell state (c_t) for long-term memory, and the hidden state (h_t) for short-term memory [14], [15]. Information flow between these states is regulated by three gates, each using a sigmoid activation to control how much data passes through (0–100%) [15].

At each timestep t, the forget gate (f_t) determines what to discard from the previous cell state by combining the input x_t with h_{t-1}. The input gate (i_t) then decides what new information to store, scaling a candidate vector $\hat{c}_t$ generated by a tanh layer. The new cell state is computed as:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \hat{c}_t$$

Finally, the output gate (o_t) filters the cell state through tanh to produce the new hidden state:

$$h_t = o_t \cdot \tanh(c_t)$$

This gating mechanism allows LSTMs to selectively remember and forget information across many timesteps [14], [15].

3.7 Truncated Backpropagation Through Time

Training RNN's is a memory- and computation-intensive process. This is because of the standard backpropagation algorithm Backpropagation Through Time (BPTT), which creates long dependency chains for calculating gradients during training. Truncated Backpropagation Through Time (TBPTT) aims to solve this by shortening the sequence and thereby the dependency on long chains of gradients. TBPTT introduces a new hyperparameter called k that determines the length of dependency. k is the number of timesteps through which gradients are backpropagated. Thereby, the gradient updates from the loss are only calculated from k steps in chunks. [16]

TBPTT therefore becomes an approximation of BPTT, since it does not handle gradient flow over the entire sequence, but rather chunks of size k. This makes the algorithm's computation order of magnitudes faster and less memory intensive. However, the model's learning becomes directly dependent on the hyperparameter k, limiting its ability to capture complex dependencies beyond k timesteps. [16]

4 Problem Statement

Current defenses against adversarial state perturbations in RL rely on training robust policies or using separate detection mechanisms that lack temporal context and require independent training procedures.

We consider an agent receiving observations $\tilde{s}_t = s_t + \delta_t$ (where $\|\delta_t\|_\infty \leq \epsilon_{FGSM}$) that must jointly learn:
1. A policy π_θ maximizing $\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$ under attack
2. A detector $d_\phi : \mathbb{R}^{\dim(\mathcal{S})} \to [0, 1]$ that estimates attack probability from prediction residuals $\text{res}_t = \tilde{s}_t - \hat{s}_t$, where $\hat{s}_t = g_\phi(\Phi_{t-1})$ is predicted from previous LSTM features.

We investigate whether integrating detection into a recurrent control architecture improves detection performance and training efficiency by leveraging:
1. Temporal dynamics in the LSTM backbone to identify attacks as prediction residuals
2. Shared parameters trained in a single optimizer step.

5 Approach

5.1 Architecture Overview

Figure 2 below provides an abstract overview of TRIDENT, showing how environment observations are processed through a shared recurrent backbone, and thereafter passed through to the policy, value and detection modules. Figure 2 also shows the temporal structure of the algorithm, including how features from the previous timestep are reused for prediction and residual-based detection. The following subsections describe each component in depth, with the complete algorithm specification provided in Algorithm 1 (see Appendix A.2).

5.2 TRIDENT

TRIDENT is built around a shared-backbone architecture for deep robust reinforcement learning (DRRL): a single LSTM backbone produces temporal representations that feed three task-specific heads, namely an actor for policy optimization, a critic for value estimation, and a predictive detection head for identifying adversarial perturbations.

The shared backbone structure offers two key advantages in DRRL: (1) parameter efficiency through weight sharing across all three tasks, and (2) the detection head is expected to benefit from features already optimized for control, rather than learning observation representations from scratch.

Unlike prior work that trains separate networks and policies for different perturbations [18], our architecture uses a single LSTM backbone that supports multi-task learning for policy optimization, value estimation, and adversarial observation prediction.

For FGSM attack detection, we introduce a predictive detection mechanism inspired by the one used in [6]. This mechanism is based on the idea that clean observations follow predictable environment dynamics, while adversarial perturbations violate these learned patterns. Our prediction head consists of two main components: (1) an observation predictor g_ϕ that maps LSTM features to predicted next observations, and (2) a residual analyzer d_ϕ that classifies an attack likelihood based on prediction errors. In our implementation, given the features Φ_t from the LSTM at time t, the predictor generates $\hat{s}_t = g_\phi(\Phi_{t-1})$ using features from the previous timestep, and the residual analyzer computes the attack probability from the prediction residual: $\text{res}_t = s_t - \hat{s}_t$.

5.3 Training Procedure

Training proceeds through a joint optimization process of the four objectives: the PPO policy objective, value function loss, observation prediction loss, and detection loss, which is a combination of focal and prediction loss (see Algorithm 1 for the complete algorithm). The

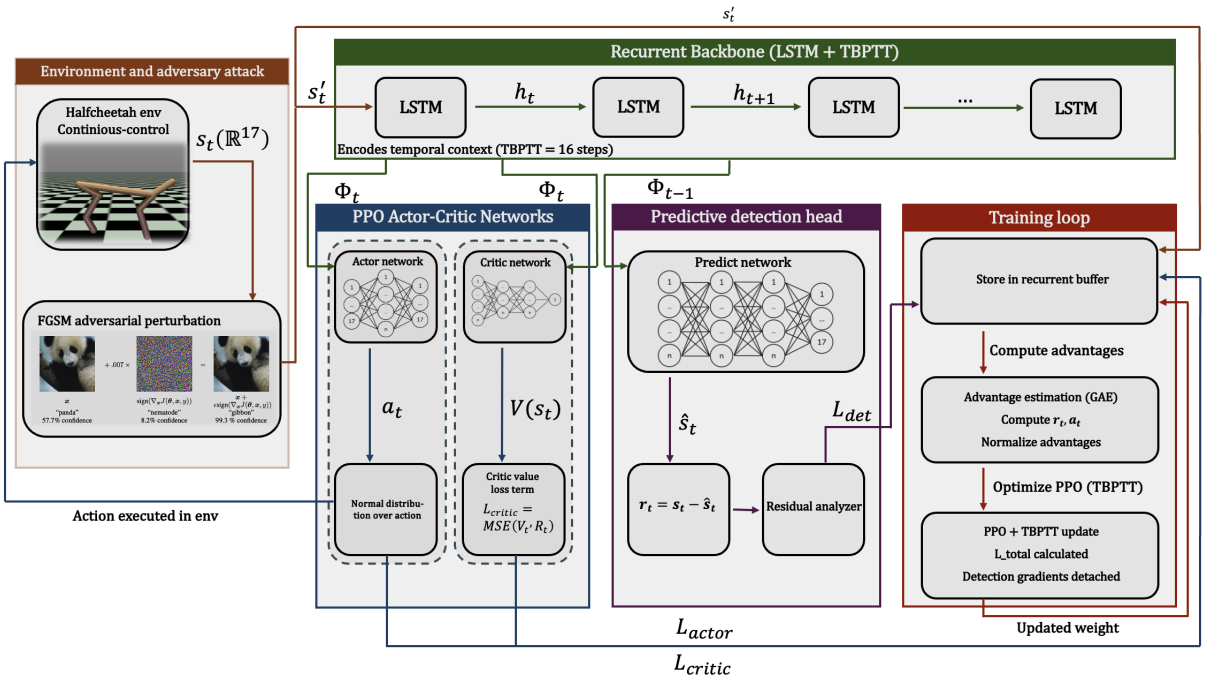

Figure 2: Architecture overview. Images in the "Environment and adversary attack" section is taken from [4], [17]

total loss is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{PPO}} + \lambda_v \mathcal{L}_{\text{val}} + \lambda_d(\mathcal{L}_{\text{focal}} + 0.1 \cdot \mathcal{L}_{\text{pred}}),$$

where λ_d & λ_v are weighting coefficients for detection and value loss.

The multi-objective loss function combines PPO, value estimation, and detection objectives in a single optimization step. This joint formulation allows the agent to learn shared representations that support both control and perturbation detection.

During rollout collection, we apply FGSM attacks to the observation with a probability of p_{attack}, where attack gradients are computed as mentioned in 3.5. We store both clean and attacked observations with labels. Training on all observations, the prediction head's goal is to produce smaller residuals for clean states than attacked states, enabling the residual analyzer to detect attacks via error magnitude. The labels are used to enable supervised learning for the detection head.

In our predictive detection head, we compute losses over a time-ordered sequence using TBPTT. At each timestep t, we use features from the previous step Φ_{t-1} to predict the current observation $\hat{s}_t$. The prediction loss is calculated using mean squared error (MSE) across all timesteps: $\mathcal{L}_{\text{pred}} = \text{MSE}(\hat{s}_t, s_t)$.

The detection loss uses focal loss and Binary cross-entropy (BCE) to handle class imbalance. BCE is defined as: $\text{BCE}(y_t, p_t) = -[y_t \log(p_t) + (1 - y_t) \log(1 - p_t)]$, which measures the discrepancy between predicted probability p_t and true label y_t. Focal loss extends BCE by down-weighting easy examples and focusing on hard-to-classify cases: $\mathcal{L}_{\text{focal}} = -\alpha_t(1 - p_t)^\gamma \cdot \text{BCE}(y_t, p_t)$, where p_t is the predicted probability of the correct class, α_t is the class-dependent weighting factor that balances positive/negative examples, and γ controls the focusing strength. Having a higher γ increases focus on hard examples.

As seen in $\mathcal{L}_{\text{total}}$, $\mathcal{L}_{\text{pred}}$ is weighted by 0.1 relative to $\mathcal{L}_{\text{focal}}$, reflecting its role as an auxiliary objective that improves residual signal quality

without dominating the primary detection task. To prevent detection training from interfering with the primary RL objectives, features are detached when computing detection losses, allowing independent gradient flow for the detection head. The model is optimized using Adam [19] with gradient clipping, and observation normalization is employed to stabilize training in continuous control environments.

5.4 Inference

We perform inference with the trained recurrent agent in a forward-only manner i.e. no gradient updates are performed. At each timestep the observation s_t is normalized and passed through the LSTM backbone to produce the features Φ_t and an updated hidden state. The actor outputs a deterministic action using the policy mean μ ($\tanh(\mu)$) and the critic provides a value estimate for logging.

For detection, the predictive head uses the features from the previous timestep Φ_{t-1} to predict the current observation $\hat{s}_t$. When the actual observation s_t is returned, the two observations are both fed to the residual analyzer which outputs the attack probability. The hidden states are reset both at the start and at the end of each episode, then propagated between timesteps. Detection outputs are logged and evaluated, but are not used to alter the agent's actions during evaluation.

6 Benchmarks

6.1 HalfCheetah-v5 Environment

HalfCheetah-v5 [17] is a continuous control benchmark from the MuJoCo physics simulator featuring a planar two-legged robot. The observation space is 17-dimensional, including joint angles, angular velocities, and positional information. The action space is 6-dimensional, controlling torques applied to six joints. The reward function encourages forward velocity while penalizing control effort, creating a challenging locomotion task where small perturbations to observations can significantly degrade performance.

6.2 Attack Epsilon Robustness Analysis

To identify the detection threshold, we trained models on FGSM attacks at epsilon values ranging from $\varepsilon = 0.01$ to $\varepsilon = 0.20$ in increments of 0.01, and evaluated each at its training epsilon. Detection accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives.

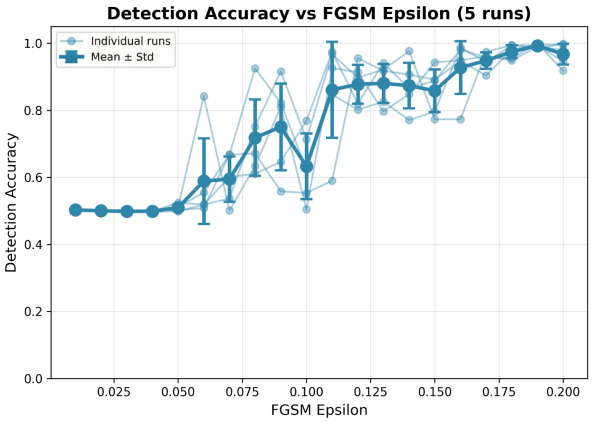

Figure 3: Attack Epsilon threshold benchmark

Figure 3 shows the performance of five individually trained models with identical hyperparameters, displaying the mean and standard deviation across all attack epsilon values. Detection accuracy remains high for $\varepsilon \geq 0.11$, maintaining above 80% accuracy. Between $\varepsilon = 0.06$ and $\varepsilon = 0.11$, accuracy falls below 80% but remains better than random guessing. Below $\varepsilon = 0.06$, performance drops to approximately 50%, indicating that the detector performs no better than random guessing. This establishes $\varepsilon = 0.11$ as the threshold for reliable detection.

6.3 Benchmark

This benchmark will evaluate our agent's resilience and baseline performance in HalfCheetah-v5 under the following conditions: a clean environment and one with FGSM attacks (The attacks will have the same ϵ_{FGSM} and probability as during training). As a point of reference, we train a StableBaselines3 (SB3) PPO agent [20], [21] with their own optimized parameters and compare the mean episode return, stability and degradation under identical attack settings. This benchmark will quantify how much performance the attacks remove, how effectively our detection preserves return, and where our method stands relative to a PPO baseline.

6.4 Comparison Study

As seen in section 6.4 below, in the attacked setting, our adversarially trained model shows robustness and consistent performance. It achieves an average reward of 3963.06 ± 110.87 over 200 episodes, with a best of 4188.18, worst of 3631.93, and a 97.53% detection accuracy. In contrast, the SB3 model collapses under the same FGSM conditions, averaging 712.66 ± 539.22, with far greater variability and occasionally negative returns, with a best reward of 3544.61 and lowest of -182.91. Overall our model delivers $+3250.4$ higher average return, a 456.09% increase.

In the clean setting, the trade-off becomes more visible. The SB3 model prevails with a average reward of 6395.84 ± 2958.42, reaching a best of 8764.25 and lowest of -147.35, whereas our adversarially model averages 4240.64 ± 63.63 (best 4393.19; lowest 4001.16), with a detection accuracy of 99.80%. Although our model is highly stable compared to the SB3 model, it underperforms by -2155.20 (-33.7%). These numbers suggest that our model prioritizes defense and robustness at the cost of peak performance when no attacks are present.

In conclusion, the comparison study demonstrates that our approach improves resilience to adversarial perturbations in maintaining high returns and tight deviation under attack, but sacrifices performance in clean settings. The variances highlight stability versus peak return as the central trade-off of our adversarial and detection driven training.

6.5 Ablation Studies

6.5.1 Without LSTM

This study isolates the effect of the LSTM-based predictive detection mechanism. We compare agents using the full TRIDENT architecture (LSTM backbone with prediction-observation-residual detection) against a feed-forward (FF) variant where the LSTM and predictive detection head are removed, and observations are fed directly into a detection head with sigmoid activation to classify attack presence. We will compare the stability and final performance under the same settings to assess how much or little LSTM contributes to robustness and policy learning. This will reveal whether removing recurrence degrades the robustness of TRIDENT or reduces peak reward, essentially clarifying the backbone's role.

To ensure a fair comparison, 5 training runs are conducted for each configuration and then averaged to account for the high variance within DRL due to inherent randomness. All settings remain identical between each training run other than the LSTM backbone. In the FF variant, observations bypass the recurrent backbone and prediction mechanism, going directly to a sigmoid classifier that outputs attack probability, removing the temporal prediction-residual computation entirely. To control potential systematic drift or variation in system load the study is done in an ABAB pattern with interleaved training runs.

As seen in Table 2, the results are evaluated based on 2 primary metrics: episode reward and detection accuracy. For each trained agent, 100 evaluation episodes are executed and then the

Setting & Model		Mean ± Std	Best	Worst	Detection Acc.
Attacked:	Ours (adv. trained)	3963.06 ± 110.87	4188.18	3631.93	97.53%
Attacked:	SB3 PPO	712.66 ± 539.22	3544.61	−182.91	–
Clean:	Ours (adv. trained)	4240.64 ± 63.63	4393.19	4001.16	99.80%
Clean:	SB3 PPO	6395.84 ± 2958.42	8764.25	−147.35	–

Table 1: Clean vs. Adversarial HalfCheetah-v5 performance.

Metric	LSTM (Mean ± Std)	FF (Mean ± Std)	p-value
Average Reward	3314.73 ± 718.19	3719.76 ± 688.77	0.3893
Detection Accuracy	$98.61\% \pm 0.38\%$	$92.39\% \pm 1.39\%$	0.000011

Table 2: LSTM vs Feedforward Ablation Study Results. See table 3 for data for individual training runs.

mean reward and detection accuracy are calculated. These results are then averaged across the 5 runs per setup. In addition, Table 3 shows the data for individual training runs. See Appendix B for more details.

Based on the results of the ablation study, there is a clear difference between LSTM and FF agents in terms of performance. The LSTM agents achieved 98.61% detection accuracy with very low variance, which significantly outperformed the FF agents (92.39%). The low p-value ($p < 0.001$) confirms that the results are statistically significant and outlines the importance of recurrence for detection accuracy. In terms of average reward, the results showed a minimal difference between LSTM agents (3314.73) compared to FF agents (3719.76). While the feedforward architecture results in a higher mean, the p-value of 0.3893 indicates that it is not statistically significant due to a high variance in both results. The average reward is therefore statistically comparable and the LSTM architecture does not suffer from a worse average reward, while achieving a higher precision in detection.

6.5.2 With and Without Detection

This study compares the proposed architecture with and without the detection head to evaluate whether the detection mechanism negatively impacts control performance. Both configurations were trained for 4000 iterations under identical conditions across 20 independent runs, with each model evaluated for 100 episodes.

The model without detection achieved a mean final 100-episode reward of 3704.90 ± 968.55, compared to 3214.44 ± 1460.07 for the model with detection. The difference of 490.46 in mean reward (13.24% lower with detection) is not statistically significant ($p = 0.2299$), indicating that while the detection head may reduce final performance, the difference is not substantial, and may be different if the trainings were run again.

The model with detection exhibited higher variance in final reward performance (1460.07 vs 968.55), suggesting potential training instability. Additionally, the model without detection achieved higher best-case performance (3942.49 vs 3530.47 mean best reward) and higher worst-case performance (489.89 vs -28.84 minimum reward). These findings suggest that the detection mechanism provides effective attack identification capability without substantially degrading overall task performance, though it may introduce some variability in performance outcomes.

7 Limitations

Our approach faces several limitations. First the predictive detection head introduces a computational overhead since each timestep requires a forward pass to produce the expected next observation and a residual calculation for comparison. Secondly treating the prediction as an auxiliary objective requires careful weighting. In our implementation the $\mathcal{L}_{pred}$ is weighted by 0.1 to keep prediction secondary to the classification signal and RL objectives. Setting this weight incorrectly can drown out the policy learning signal or leave the predictor too weak to produce a usable residual. Third, the detection quality depends on the residual threshold, meaning different thresholds may not work optimally for different attack strengths. Fourth, we have only tested with FGSM applied to observations. Because of this different attacks such as Policy Gradient Descent (PGD) may fool the prediction model. Furthermore action-space or rewards have not been tested either. Fifth, balancing three objectives: policy optimization, value estimation, and adversarial attack detection requires careful tuning. The two coefficients λ_p & λ_d are what balances this, meaning certain values may produce worse results.

8 Future Work

8.1 Correction Module

While our detection mechanism identifies FGSM perturbations, it does not include a correction module to clean observations after an attack is detected. Coupling real-time detection with real-time correction could be valuable, though this was not the focus of the current project.

8.2 Further Experiments

Several experimental directions remain for future work. Each model in our experiments was trained and benchmarked on a single attack epsilon. Training a single model on multiple epsilon values simultaneously would reveal whether the architecture can generalize across different perturbation magnitudes.

Beyond epsilon variation, we only trained and evaluated on FGSM attacks. Extending to other white-box attacks such as PGD and to black-box attacks would show how well our detection mechanism generalizes to different attack strategies.

Our evaluation was also limited to HalfCheetah-v5. Testing on other MuJoCo environments would show how different observation dimensions and dynamics affect detection accuracy and policy robustness. Adding sensor noise to observations would further help assess real-world applicability. Our residual-based detection relies on accurate observation predictions, and sensor noise may increase residuals in ways that get misclassified as attacks. Whether this causes significant false positives in practice remains unknown.

9 Conclusion

In this work we proposed a recurrent PPO-based reinforcement learning architecture that integrates a detection head based on the method used in the work by Zizzo et al. [6]. By coupling a shared LSTM backbone with an actor, critic, and detection head, the agent learns both control, value estimation, and perturbation detection within a single model. Our Benchmarks on the Halfcheetah-v5 environment show that this approach and architecture yields robustness under FGSM attack, in that the agent maintains a stable performance, while also having detection accuracies near 98%, outperforming a baseline PPO that collapses under identical adversarial conditions.

These results further revealed a meaningful performance trade-off. While our adversarially trained model shows a substantially improved stability and returns under attack, it sacrifices peak performance in clean settings where a

baseline PPO agent achieves higher rewards, but with far greater instability. The ablation studies highlight the importance of both the recurrent backbone and the detection head, showing that recurrence is essential for learning temporally consistent features, and that the detection head provides attack identification without degrading control performance, while models with detection exhibited greater training stability, though the underlying cause remains uncertain.

Overall, our findings indicate that adversarial detection can be integrated within a recurrent PPO architecture without compromising control performance, offering a promising direction for robust continuous control. Nevertheless, limitations remain regarding computational overhead, sensitivity to loss weighting, and generalization to a broader variety of attacks. Future work could address these limitations by training across multiple ϵ_{FGSM} values and evaluating in diverse environments, while also exploring a correction module to actively recover from detected attacks.

References

[1] V. Mnih et al., *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1312.5602.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1707.06347.

[3] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, *Robust adversarial reinforcement learning*, 2017. arXiv: 1703.02702 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1703.02702.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, 2015. arXiv: 1412.6572 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1412.6572.

[5] Y. Liang, Y. Sun, R. Zheng, and F. Huang, *Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning*, 2022. arXiv: 2210.05927 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2210.05927.

[6] G. Zizzo, C. Hankin, S. Maffeis, and K. Jones, "Adversarial attacks on time-series intrusion detection for industrial control systems," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 899–910. DOI: 10.1109/TrustCom50675.2020.00121.

[7] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, *Memory-based control with recurrent neural networks*, 2015. arXiv: 1512.04455 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1512.04455.

[8] Y. Qiaoben, C. Ying, X. Zhou, H. Su, J. Zhu, and B. Zhang, *Understanding adversarial attacks on observations in deep reinforcement learning*, 2023. arXiv: 2106.15860 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.15860.

[9] H. Zhang et al., *Robust deep reinforcement learning against adversarial perturbations on state observations*, 2021. arXiv: 2003.08938 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2003.08938.

[10] A. G. B. Richard S. Sutton, "Reinforcement learning: An introduction," Available at: https://shorturl.at/D9Fce, Ph.D. dissertation, Stanford University, 2015.

[11] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation*, 2018. arXiv: 1506.02438 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1506.02438.

[12] R. Kizito, P. Scruggs, X. Li, M. Devinney, J. Jansen, and R. Kress, "Long short-term memory networks for facility infrastructure failure and remaining useful life prediction," *IEEE Access*, vol. 9, pp. 67 585–67 594, 2021. DOI: 10.1109/ACCESS.2021.3077192.

[13] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*, 2013. arXiv: 1211.5063 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1211.5063.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://www.bioinf.jku.at/publications/older/2604.pdf.

[15] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023, https://D2L.ai.

[16] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation: Theory, architectures, and applications*, vol. 1, Erlbaum Hillsdale, NJ, 1995, pp. 433–486.

[17] E. Todorov, T. Erez, and Y. Tassa, *Halfcheetah-v5 mujoco environment*, Gymnasium, 2018. [Online]. Available: `https://gymnasium.farama.org/environments/mujoco/half_cheetah/`.

[18] D. K. Panda and W. Guo, *Robust policy switching for antifragile reinforcement learning for uav deconfliction in adversarial environments*, 2025. arXiv: `2506.21127 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2506.21127`.

[19] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1412.6980`.

[20] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: `http://jmlr.org/papers/v22/20-1364.html`.

[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, *Stable-baselines3: PPO implementation*, `https://github.com/DLR-RM/stable-baselines3`, Proximal Policy Optimization (PPO) implementation in Stable-Baselines3, 2021.

A Detailed Algorithm Specifications

This appendix provides comprehensive algorithmic descriptions with detailed mathematical formulations for the reinforcement learning architecture with integrated anomaly detection.

A.1 Mathematical Notation Summary

- $\mathcal{E}$: Environment
- $\mathcal{S}, \mathcal{A}, \mathcal{H}$: State space, action space, and LSTM hidden state space
- d_h: LSTM hidden dimension
- B: Batch size for mini-batch updates
- K: Number of training epochs per iteration
- N_{steps}: Number of environment steps collected per iteration
- $\pi_\theta(a|s)$: Policy parameterized by θ (outputs action distribution)
- θ_{old}: Stored parameters of old policy for PPO clipping
- $V_\theta(s)$: Value function estimating expected return from state s
- V_{clip}: Clipped value function for PPO value loss
- $f_\theta(s_t, h_{t-1})$: LSTM backbone mapping observation and previous hidden state to features and new hidden state: $(\mathcal{S} \times \mathcal{H}) \to \mathbb{R}^{d_h} \times \mathcal{H}$
- $h_t \in \mathcal{H}$: LSTM hidden state at timestep t (includes cell state)
- $\Phi_t \in \mathbb{R}^{d_h}$: Feature representation from LSTM at timestep t
- $g_\phi(\Phi_{t-1})$: Observation predictor, predicts $\hat{s}_t$ from previous features Φ_{t-1}
- $d_\phi(\text{res})$: Residual analyzer, maps prediction error to attack probability
- $\mathcal{D}$: Experience replay buffer
- $\mathcal{S}_b$: Sequence batch sampled from buffer
- s_t: State/observation at timestep t
- $\tilde{s}_t$: Attacked observation ($\tilde{s}_t = s_t + \delta_t$ where $\|\delta_t\|_\infty \leq \epsilon_{FGSM}$)
- s_{raw}, s_{norm}: Raw and normalized observations
- a_t, r_t: Action and reward at timestep t
- $\hat{s}_t$: Predicted observation from $g_\phi(\Phi_{t-1})$
- res_t: Prediction residual ($\text{res}_t = s_t - \hat{s}_t$)
- y_t: Binary attack label (0=clean, 1=attacked)
- $\hat{A}_t$: Advantage estimate using Generalized Advantage Estimation (GAE)
- $\hat{R}_t$: Return target for value function ($\hat{R}_t = \hat{A}_t + V_\theta(s_t)$)

- δ_{t+l}^{V}: Temporal difference error for GAE computation
- $r_t(\theta)$: PPO probability ratio $\pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$
- μ_A, σ_A: Mean and standard deviation of advantages (for normalization)
- T_{seq}: Sequence length for Truncated Backpropagation Through Time (TBPTT)
- ϵ_{FGSM}: FGSM attack perturbation bound
- p_{attack}: Probability of applying adversarial attack during training
- λ_d: Weight balancing detection loss relative to RL losses
- λ_v: Value function loss coefficient (embedded in c_1)
- γ: Discount factor for future rewards
- γ_{focal}: Focal loss focusing parameter (set to 2.0)
- λ_{GAE}: GAE parameter for bias-variance tradeoff
- α: Learning rate
- α_t, p_t: Focal loss class weighting terms (α_t balances pos/neg, p_t is predicted probability)
- ϵ_{clip}: PPO clipping parameter
- c_1, c_2: Value function and entropy coefficients
- g_{max}: Maximum gradient norm for clipping
- $\mathcal{L}_{\text{actor}}$, $\mathcal{L}_{\text{PPO}}$: PPO clipped policy loss with entropy bonus
- $\mathcal{L}_{\text{critic}}$, $\mathcal{L}_{\text{val}}$: Value function MSE loss with clipping
- $\mathcal{L}_{\text{focal}}$, $\mathcal{L}_{\text{det}}$: Focal loss for detection ($\gamma_{focal} = 2.0$, $\alpha = 0.25$)
- $\mathcal{L}_{\text{pred}}$: Auxiliary prediction loss (MSE between $\hat{s}_t$ and s_t)
- $\mathcal{L}_{\text{detect}}$: Combined detection loss ($\mathcal{L}_{\text{focal}} + 0.1 \cdot \mathcal{L}_{\text{pred}}$)
- $\mathcal{L}_{total}$: Total combined loss for gradient update
- $\mathcal{H}[\pi_\theta]$: Entropy of policy distribution
- detach($\cdot$): Gradient stopping operation (prevents backpropagation)

Note on Gradient Flow: The detection head receives detached LSTM features (detach($\Phi_{1:T}$)), preventing detection gradients from backpropagating through the backbone. This allows: (1) the LSTM backbone to learn RL objectives without interference from detection gradients, and (2) the detection head (g_ϕ, d_ϕ) to train on the same feature representations while learning its own parameters independently.

A.2 Algorithm 1: Recurrent PPO with Predictive Detection

Implementation Notes:

Algorithm 1 Recurrent PPO with Predictive Detection (LSTM + TBPTT)

Require: $\mathcal{E}$, d_h, T_{seq}, $(\epsilon_{FGSM}, p_{attack})$, $(\epsilon_{clip}, c_1, c_2, \gamma, \lambda_{GAE})$, λ_d, α, B, K
Ensure: Trained LSTM policy π_θ with predictive detection head

Initialize f_θ, π_θ, V_θ, g_ϕ, d_ϕ
for iteration $i = 1, 2, \ldots$ **do**
  $\theta_{old} \leftarrow \theta$; $\mathcal{D} \leftarrow \emptyset$; $s_0 \sim \mathcal{E}$, $h_0 \leftarrow \mathbf{0}$
  for $t = 0$ to N_{steps} **do**
    $s_{raw} \leftarrow s_t$, $y_t \leftarrow 0$
    if rand() $< p_{attack}$ **then**
      $g \leftarrow \nabla_{s_{raw}}(-V_\theta(s_{raw}))/(\sqrt{\mathrm{Var}[s]} + \epsilon)$ ▷ Gradient in raw space
      $s_{raw} \leftarrow s_{raw} + \epsilon_{FGSM} \cdot \mathrm{sign}(g)$; $y_t \leftarrow 1$
    end if
    $s_{norm} \leftarrow \mathrm{norm}(s_{raw})$; $(\Phi_t, h_t) \leftarrow f_\theta(s_{norm}, h_{t-1})$; $a_t \sim \pi_{\theta_{old}}(\Phi_t)$, $v_t \leftarrow V_\theta(\Phi_t)$
    Execute a_t, observe $s_{t+1}, r_t, \mathrm{done}_t$
    Store $(s_{norm}, a_t, r_t, v_t, y_t, \mathrm{done}_t)$ in $\mathcal{D}$
    if done_t **then** $s_{t+1} \sim \mathcal{E}$, $h_{t+1} \leftarrow \mathbf{0}$
    end if
  end for
  Compute GAE: $\hat{A}_t, \hat{R}_t$; Normalize: $\hat{A}_t \leftarrow (\hat{A}_t - \mu_A)/(\sigma_A + \epsilon)$
  for epoch $k = 1, \ldots, K$ **do**
    for each batch $\mathcal{S}_b$ of size B, length T_{seq} **do**
      $(\Phi_{1:T}, h_T) \leftarrow f_\theta(s_{1:T}, \mathbf{0})$ ▷ TBPTT: fresh hidden state per sequence
      $\mathcal{L}_{\text{actor}} = -\mathbb{E}[\min(r_t \hat{A}_t, \mathrm{clip}(r_t, 1 \pm \epsilon_{clip})\hat{A}t)] - c_2 \mathcal{H}[\pi\theta]$
      $\mathcal{L}_{\text{critic}} = c_1 \mathbb{E}[\max((V_\theta - \hat{R}_t)^2, (V_{clip} - \hat{R}_t)^2)]$
      $\hat{s}_t \leftarrow g_\phi(\mathrm{detach}(\Phi_{t-1}))$; $\mathrm{res}_t \leftarrow s_t - \hat{s}_t$ ▷ Detached features
      $\mathcal{L}_{\text{focal}} = \mathbb{E}_t[\alpha_t (1 - p_t)^{\gamma_{focal}} \cdot \mathrm{BCE}(d_\phi(\mathrm{res}_t), y_t)]$ ▷ $\alpha = 0.25$, $\gamma_{focal} = 2.0$
      $\mathcal{L}_{\text{pred}} = 0.1 \cdot \mathbb{E}_t[\|\hat{s}_t - s_t\|^2]$ ▷ Auxiliary prediction loss
      $\mathcal{L}_{\text{detect}} = \mathcal{L}_{\text{focal}} + \mathcal{L}_{\text{pred}}$
      $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{actor}} + \mathcal{L}_{\text{critic}} + \lambda_d \mathcal{L}_{\text{detect}}$
      Update: $(\theta, \phi) \leftarrow (\theta, \phi) - \alpha \cdot \mathrm{clip}(\nabla \mathcal{L}_{total}, g_{max})$
    end for
  end for
  Step LR scheduler
end for
return $\pi_\theta, f_\theta, g_\phi, d_\phi$

1. *FGSM Attack Placement:* Attacks are applied in raw observation space before normalization (lines 7-8) to preserve the attack signal. The gradient is computed w.r.t. normalized observations, then scaled back to raw space by dividing by $\sqrt{\text{Var}[s]}$.

2. *Feature Detachment:* Features are detached before passing to the detection head (line 22), preventing detection gradients from interfering with the LSTM backbone's RL learning while allowing the detection head (g_ϕ, d_ϕ) to train independently.

3. *Focal Loss:* Detection uses focal loss with $\gamma_{focal} = 2.0$, $\alpha = 0.25$ (line 23) to address class imbalance between clean and attacked observations.

4. *Auxiliary Prediction:* The prediction loss (line 24) with weight 0.1 trains the predictor g_ϕ to anticipate observations, enabling detection via prediction residuals.

5. *TBPTT:* Each sequence batch starts with fresh hidden state $\mathbf{0}$ (line 13), enabling parallel sequence processing during training.

B Without LSTM Ablation Results

This appendix details the complete experimental results from the LSTM ablation study. To account for the variable nature of DRL and to document the experiment, we present the raw performance data for each of the five independent training runs. Table 3 lists the specific Average Reward and Detection Accuracy values for both the LSTM and Feedforward (FF) architectures across all runs, serving as the basis for the statistical means reported in the analysis. The used hyperparameters are from the benchmark trial with 256 hidden dimensions and FGSM attacks applied with epsilon magnitude of 0.15 at a 30% probability.

Table 3: Complete Training Run Data: LSTM vs Feedforward (FF)

Setting	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**
LSTM (Reward)	3990.09	3737.74	3258.37	2129.63	3457.83
LSTM (Det. Acc.)	99.16%	98.61%	98.57%	98.10%	98.60%
FF (Reward)	4293.51	4075.16	2831.29	4270.34	3128.49
FF (Det. Acc.)	92.61%	90.59%	94.32%	92.79%	91.65%