Evaluating and Enhancing the Robustness of Proximal Policy Optimization to Test-Time Corruptions in Sequential Domains

Mate Rodić[1]
Supervisor(s): Frans Oliehoek[1]**, Mustafa Celikok**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Mate Rodić
Final project course: CSE3000 Research Project
Thesis committee: Frans Oliehoek, Mustafa Celikok, Annibale Panichella

Abstract

Reinforcement learning (RL) agents often achieve impressive results in simulation but can fail catastrophically when facing small deviations at deployment time. In this work, we examine the *brittleness* of Proximal Policy Optimization (PPO) agents when subjected to test-time observation noise and we evaluate techniques for improving *robustness*. We compare four variants: feed-forward PPO, Recurrent PPO (with LSTM memory), Noisy-PPO (trained with injected observation noise), and Recurrent-Noisy PPO, across two benchmarks: the classic CartPole-v1 and the more realistic Highway-env. Performance is measured over 100 episodes per corruption level, using mean return, success rate, and the Area-Under-Degradation-Curve (AUDC) as robustness metrics. Our results show that recurrent-noise-augmented training yields the largest gains, as Noisy-PPO maintains its clean performance at high noise levels, while recurrence alone offers modest improvements. In the highway environment, both training noise injection and LSTM memory yield improved returns across both environments, indicating that a simple integration of noise augmentation or recurrence can, to some extent, improve PPO's robustness to real-world uncertainties.

1 Introduction

Reinforcement learning (RL) has become a leading approach for sequential decision-making, with successes in finance, healthcare, and game-playing [13]. Yet agents that perform well in simulation often fail when realistic deviations occur at test time. We call such a sudden collapse in performance *brittleness*. In contrast, *robustness* is the ability of an agent to continue operating effectively despite minor perturbations, such as sensor noise or control delays.

In real-world applications, such as robotic manipulation, personalized recommendation, or financial trading, even slight corruptions (e.g. Gaussian observation noise, delayed actions, or adversarial inputs) can cause unexpected failures. Ensuring robustness under these conditions is critical for safe and reliable deployment.

We evaluate robustness on two standard benchmarks. **CartPole-v1** is a simple control task with a four-dimensional state (pole angle, cart position, etc.) and two discrete actions; the goal is to balance a pole on a cart for as long as possible [1]. **Highway-env** is a richer driving scenario with continuous observations (vehicle positions, speeds, lane indicators) and a discrete action set (accelerate, brake, lane-change); it tests an agent's ability to navigate multi-lane traffic safely [7].

Previous work on improving the robustness of RL agents has used methods like adversarial training [9], domain randomization, and data augmentation [8]. These approaches help agents generalize better, but they rarely test how policies hold up against precise, controlled perturbations at deployment time.

Proximal Policy Optimization (PPO), proposed by Schulman et al. [12], is a popular on-policy algorithm because it keeps each policy update small and reliable, leading to stable learning. An extension called Recurrent PPO [10] adds an LSTM layer, a type of neural network unit that can remember information over time, to help the agent handle environments where not all state information is visible at once.

Despite these advances, we still lack a head-to-head comparison of how much memory (via LSTMs) or training-time noise injection each helps reduce an agent's brittleness when it faces real-world corruptions like sensor noise or delayed actions.

This research aims to fill that gap, investigating how small, controlled test-time corruptions, specifically Gaussian observation noise, because it closely models real sensor error and environmental randomness, affect standard PPO agents in classic Gym environments (CartPole-v1 and Highway-env). We evaluate whether enhancements such as recurrent architectures (Recurrent PPO) or explicit training-time noise injection (Noisy-PPO), or a combination of the two (Noisy Recurrent PPO), can effectively counteract performance degradation.

More specifically, we ask: (1) How does standard PPO performance degrade as test-time perturbations increase? (2) To what extent can recurrent architectures or noise-augmented training mitigate this degradation? We provide systematic experimental evaluations, comparisons among PPO, Recurrent PPO, and Noisy variations of PPO and recurrent PPO, and statistical analyses of results.

The remainder of the paper is structured as follows: Section 2 describes in more detail the work related to our project, the following section details our methodology of the research, Section 4 presents experimental results and analysis, Section 5 concludes our research, while Section 6 discusses responsible research considerations. Appendix includes additional remarks about LLM usage and execution of training jobs.

2 Related Work

Many researchers have looked at making RL agents more robust to uncertainty and adversarial interference. Pinto et al. [9] frame a minimax game where an adversary perturbs observations or actions, and show that adversarial training produces policies that handle worst-case disturbances better. Domain randomization methods [15] train agents on a variety of simulated variations, such as physics parameters or sensor noise, so they transfer more reliably to real environments. Moos et al. [8] study data augmentation in continuous-control tasks by adding noise to observations during training to boost generalization.

Proximal Policy Optimization (PPO), introduced by Schulman et al. [12], is widely used because it uses a clipped objective to keep each policy update small and stable. However, the original PPO assumes perfect sensors and no action delay. To cope with partial observability, Pleines et al. [10] add an LSTM layer to PPO and report small gains in hidden-state tasks, but they do not test noise robustness. NoisyNet [2] injects parameter noise to encourage exploration, yet it does not address corruption at test time.

More recent work directly evaluates test-time robustness. For example, Zhang et al. [16] measure how policies degrade under Gaussian and adversarial noise and find that training with noise helps, and Huang and et al. [6] introduce a noise curriculum during training to improve stability. These studies, however, usually consider only one type of perturbation or one environment.

Our study aims to fulfill this void by systematically testing PPO under a controlled corruption (Gaussian observation noise) by comparing four variants (standard PPO, Recurrent PPO, Noisy-PPO, and Recurrent-Noisy PPO) across two gym environments, using paired statistical tests to isolate the benefits of memory versus noise injection.

3 Methodology

This section describes our experimental framework for assessing the robustness of Proximal Policy Optimization (PPO) variants under controlled test-time corruptions. We begin with an overview of the environments, then present the four algorithmic configurations, detail the corruption mechanisms, and finally define our evaluation metrics and statistical tests.

3.1 PPO Algorithm and it's Variants

Proximal Policy Optimization (PPO) [12] is an on-policy, policy-gradient algorithm engineered for stable and efficient training. As an on-policy method, it learns from data generated by the most recent version of the policy. The core of PPO's stability lies in its distinctive clipped surrogate objective function, which prevents destructive, large-scale policy updates.

After collecting a batch of trajectories with the current policy, PPO optimizes the following objective:

$$L(\theta) \;=\; E_t\Big[\min\big(r_t(\theta)\,A_t,\;\mathrm{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\,A_t\big)\Big],$$

Here, $r_t(\theta)$ represents the probability ratio between the new and old policies, A_t is the estimated advantage of an action, and ϵ is a hyperparameter defining the clipping range. By clipping the ratio $r_t(\theta)$, the objective function discourages policy updates that deviate too far from the previous policy, effectively enforcing a trust region constraint without the computational overhead of second-order methods.

Rather than implementing PPO from scratch, our experiments are built upon the stable-baselines3 library (v2.2.1), as well as sb3-contrib (v2.2.1) in case of recurrent PPO, a standard for reproducible reinforcement learning research. Our baseline configuration uses the library's default PPO agent, meaning that for all agents, we used a learning rate of x=0.0003, a discount factor of 0.99, a GAE lambda of 0.95, and a clip range of 0.2. Our noise-augmented versions of PPO algorithms are built upon the mentioned baselines, using a noise-injection wrapper. By using these already well-tested implementations, we ensure our focus remains on the comparative performance of these distinct architectures.

To isolate the effects of memory and noise-based training, we compare four PPO-based configurations:

3.1.1 Feed-forward PPO

The baseline uses a multilayer perceptron for both policy and value networks, following the original PPO formulation [12]. The network architecture consists of two hidden layers with 64 units each. No recurrence or noise injection is applied during training.

3.1.2 Recurrent PPO

Recurrent PPO extends the feed-forward policy by inserting a Long Short-Term Memory (LSTM) layer after the final dense layer. An LSTM is a type of recurrent neural network that maintains an internal hidden state and a cell state, controlled by input, forget, and output gates. This gating mechanism allows the agent to selectively retain or discard information over long time horizons, which is crucial in partially observable settings where the full environment state is not available at each step.

In practice, the LSTM processes the sequence of past observations and actions, enabling the policy to infer unobserved variables (e.g. object velocities or other agents' intents) from temporal patterns. For example, in highway-style driving, the LSTM can learn to remember a nearby vehicle's recent speed changes, improving lane-change decisions. During training, the hidden and cell states are reset at episode boundaries, ensuring that memory is used only within each episode.

3.1.3 Noisy-PPO

The following configuration injects Gaussian noise into state observations during training. At each step, the raw state vector is perturbed by adding noise $\mathcal{N}(0, \sigma_{\text{train}}^2, \mathrm{I})$, where the noise standard deviation σ_{train} was held constant at [e.g., 0.1]. This encourages the learned policy to generalize to noisy inputs. No noise is applied during evaluation.

3.1.4 Recurrent-Noisy PPO

This configuration combines the LSTM architecture of Recurrent PPO with the same training-time noise injection used in Observation-Noise PPO, aiming to capture both memory-based and noise-based robustness benefits.

3.2 Gaussian State Noise

When evaluating the robustness of our trained policies, we subjected all four PPO configurations to controlled levels of Gaussian noise during their evaluation phase. This protocol is distinct from the noise-injection used to train the `Noisy-PPO` variants and was applied to every agent to ensure a fair comparison.

At each timestep t of an evaluation episode, the agent receives a perturbed state observation $\hat{\mathbf{s}}_t$ instead of the true state $\mathbf{s}_t$. The perturbation is achieved by adding a noise vector $\mathbf{z}_t$ where each component is drawn independently from a zero-mean Gaussian distribution:

$$\hat{\mathbf{s}}_t = \mathbf{s}_t + \mathbf{z}_t, \quad \text{where} \quad \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$$

We performed a sweep of evaluations using a range of noise intensities. The standard deviation σ was selected from the set $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. The case where $\sigma = 0.0$ corresponds to a noise-free evaluation and serves as the baseline metric. This methodology simulates real-world sensor inaccuracies or environmental randomness, allowing us to measure the degradation in performance as those inaccuracies increase.

3.3 Environments

To cover both simple control and more realistic driving scenarios we use two OpenAI Gym benchmarks.

3.3.1 CartPole-v1

The CartPole environment provides a four-dimensional state (cart position, cart velocity, pole angle, pole velocity at tip) and two actions (move left or right). The agent's goal is to balance the pole on the cart for as long as possible [1].

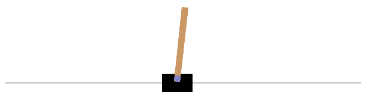

Figure 1: Snapshot of the CartPole-v1 environment.

3.3.2 Highway-env

Highway-env simulates highway driving with continuous observations (vehicle positions, speeds, lane indices) and a discrete action set (accelerate, brake, lane-change left/right) [7]. In this environment, agents must navigate multi-lane traffic safely, avoiding collisions and maintaining speed.

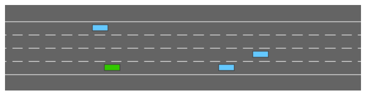

Figure 2: Example scenario in Highway-env.

3.4 Hyperparameter Tuning

To ensure better agent performance, we conducted a hyperparameter search for the two parameters introduced in our experimental configurations: the LSTM hidden state size and the training noise magnitude. The performance of each parameter value was evaluated by training three independent agents and averaging their mean return over 20 evaluation episodes in a noise-free environment.

3.4.1 LSTM Hidden State Size

The best hidden state size for the Long Short-Term Memory (LSTM) layer was determined by tuning the `Recurrent PPO` agent. We evaluated a range of potential sizes for the LSTM's hidden state vector. The values included in our search were

$$\{5, 10, 15, 20, 25, 30\}.$$

Following the results of this search are presented in Figure 3, we continued using an LSTM size of 10 in our subsequent trainings.

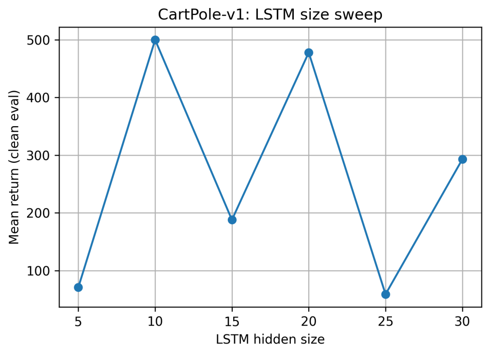

Figure 3: Mean evaluation return of the `Recurrent PPO` agent as a function of the LSTM hidden state size. Ran on the Cartpole environment for 50,000 timesteps.

3.4.2 Training Noise Magnitude (σ_{train})

To find the appropriate level of noise for regularizing the policy during training, we tuned the standard deviation of the Gaussian noise, σ_{train}, using the `Observation-Noise PPO` agent. We swept through values of σ :

$$\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}.$$

As shown in Figure 4, a moderate amount of noise proved beneficial. The agent's performance was maximized at a σ_{train} value of **0.1**. Both lower and higher levels of noise resulted in weaker performance.

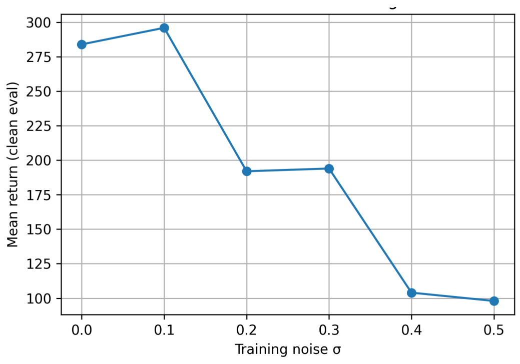

Figure 4: Mean evaluation return of the `Noisy PPO` agent as a function of the training noise standard deviation, σ_{train}. Error bars represent the standard deviation over three independent seeds.

3.5 Training Configuration

Based on our hyperparameter search, we selected an LSTM hidden state size of **10** and a training noise standard deviation σ_{train} of **0.1**. These values were used to configure the `Recurrent PPO` and `Recurrent-Noisy PPO` agents for their final training runs, ensuring a fair and optimized comparison against the baseline. All agents were trained for 100,000 time steps in both environments, Cartpole-v1 and highway-fast-v0, using 5 randomly chosen seeds (42, 123, 587, 1026, and 2025) to capture variability between runs. The budget of 100,000 steps was chosen based on pilot runs over Cartpole-v1 environment showing that it is sufficient for stable convergence on both tasks without excessive computation.

For the rest of the parameters, we adopt common defaults that yielded a satisfactory compromise between performance of agents and time efficiency, which was estimated through our preliminary testing and familiarizing with the environments and agents.

For the feed-forward PPO baseline, those are:

$$n_{\text{steps}} = 2048, \quad \text{batch\_size} = 128,$$

$$n_{\text{epochs}} = 4, \quad \alpha = 3 \times 10^{-4}, \quad \epsilon = 0.2.$$

, where α is number learning rate and ϵ is clip range. These settings balance update frequency and sample utilization, providing reliable performance across both environments.

To train recurrent agents efficiently, we found out that increasing the number of steps to $n_{\text{steps}} = 512$ increases time performance of the agent. Different to baseline PPO, we use 10 epochs and use two fully-connected layers of 64 units before the LSTM.

As previously mentioned, for Noisy-PPO and Recurrent-Noisy PPO, we inject zero-mean Gaussian noise with $\sigma_{\text{train}} = 0.2$ into each observation at training time, while other parameters are the same as for other algorithm configurations.

3.6 Model Evaluation

Each trained model is evaluated under every corruption level σ by running $N = 100$ episodes. For each tuple $(\text{env}, \text{seed}, \text{agent}, \sigma)$, let R_i be the return from episode i. We compute:

Mean return:

$$\mu(\sigma) = \frac{1}{N} \sum_{i=1}^{N} R_i,$$

Standard deviation:

$$\sigma_r(\sigma) = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \big(R_i - \mu(\sigma)\big)^2}.$$

Here, $\mu(\sigma)$ quantifies average task success under corruption level σ, and $\sigma_r(\sigma)$ captures performance consistency across trials.

To summarize robustness in a single scalar, we define the normalized Area-Under-Degradation-Curve (AUDC) as

$$\text{AUDC} = \frac{\displaystyle\int_{\sigma_{\min}}^{\sigma_{\max}} \mu(\sigma)\, \mathrm{d}\sigma}{(\sigma_{\max} - \sigma_{\min})\, \mu(\sigma_{\min})},$$

where $\sigma_{\min} = 0$ and $\sigma_{\max} = 0.5$. In practice, with M equally spaced noise levels $\{\sigma_j\}$, we approximate

$$\text{AUDC} \approx \frac{1}{M\, \mu(0)} \sum_{j=1}^{M} \mu(\sigma_j)\, \Delta\sigma,$$

with $\Delta\sigma = (\sigma_{\max} - \sigma_{\min})/(M-1)$. A larger AUDC indicates that an agent preserves a greater fraction of its clean performance as noise increases.

Finally, to test for statistically significant differences between each PPO variant and the feed-forward PPO baseline, we perform paired t-tests at significance level $\alpha = 0.05$. Returns are paired by matching runs with identical seed and corruption level, which controls for random variation and isolates the effect of the algorithmic modification on robustness.

4 Results

In the following section, we present our findings on how the four PPO variants perform under different amounts of Gaussian observation noise.

For clarity, and to ensure the reproducibility and consistence of the results, every result presented here uses the same training configuration, as mentioned in more detail in the previous section (100,000 timesteps, 5 random seeds). The further description and contents of the data and code are available in the Appendix.

4.1 Performance under Gaussian Observation Noise

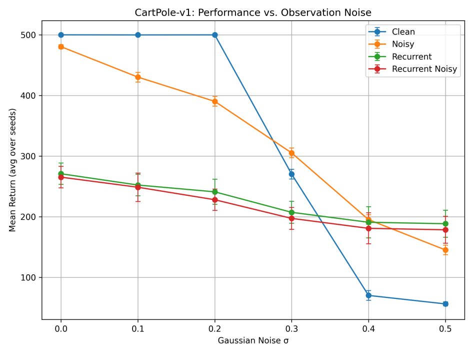

Figure 5: Mean return vs. Gaussian noise σ on CartPole-v1 for each PPO variant.

Figure 5 shows that standard PPO's mean return on CartPole-v1 falls steeply as σ increases: from nearly 500 at $\sigma = 0.0$ down to about 160 at $\sigma = 0.5$. Adding an LSTM (Recurrent PPO) reduces the slope of degradation but lowers the initial return. Recurrent and recurrent noisy variants give lower returns at smaller Gaussian noise values, however, they hold onto better than the other two when the noise amount increases.

Figure 6 illustrates similar trends on Highway-env: standard PPO drops from about 30 to 10 as σ increases, Recurrent PPO flattens the decline modestly but at a lower starting point, Noisy-PPO retains over 18 return at $\sigma = 0.5$, and Recurrent-Noisy PPO falls just above it.

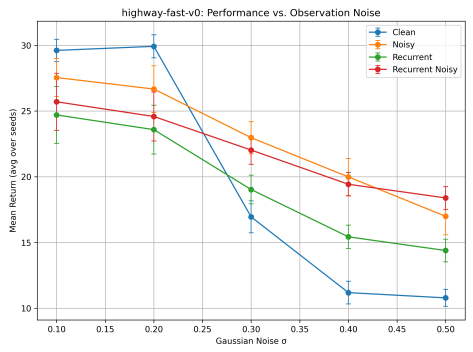

Figure 6: Mean return vs. Gaussian noise σ on Highway-env for each PPO variant. Noisy-PPO again achieves the best robustness, with Recurrent-Noisy PPO close behind. Standard PPO degrades fastest and Recurrent PPO offers limited improvement.

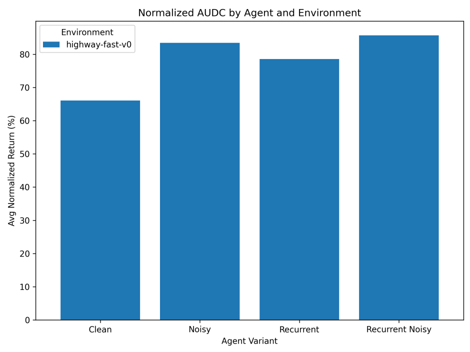

Figure 8: Normalized Area-Under-Degradation-Curve (AUDC) on Highway-env for each PPO variant.

4.2 Aggregate Robustness and Statistical Analysis

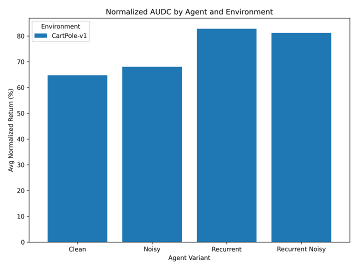

Figure 7: Normalized Area-Under-Degradation-Curve (AUDC) on CartPole-v1 for each PPO variant. Higher values indicate better robustness to Gaussian noise and action delay.

Figure 7 shows that Recurrent model achieves the highest AUDC on CartPole, followed closely by Recurrent-Noisy PPO (0.80). Noisy PPO improves over the feed-forward baseline, but remains below the noise-augmented variants.

As seen in Figure 8, the Highway-env results follow a similar pattern, with a clearer improvement over the baseline for all the agents. Recurrent-Noisy-PPO leads with, while both Noisy and Recurrent variations score visibly above the baseline PPO's relatively low performance. These aggregate metrics confirm that training-time noise injection with combination of LSTM recurrence delivers the most significant robustness gains across both environments.

4.3 Statistical significance via paired t-tests

We ran two-sided paired t-tests ($\alpha = 0.05$) on the per-seed mean returns of Recurrent-PPO versus clean-PPO at each noise level σ. Tables 1 and 2 list the mean difference $\Delta\bar{r}$, t-statistic and p-value. Any $p < 0.05$ denotes statistical significance.

Table 1: CartPole-v1: Paired t-test results for Recurrent-PPO vs. clean-PPO.

σ	$\Delta\bar{r}$	t	p
0.0	-229.12	29.97	< 0.001
0.1	-247.58	31.55	< 0.001
0.2	-258.87	27.60	< 0.001
0.3	-62.98	6.42	0.003
0.4	120.62	-9.33	0.001
0.5	132.22	-13.76	< 0.001

Table 2: Highway: Paired t-test results for Recurrent-PPO vs. clean-PPO.

σ	$\Delta\bar{r}$	t	p
0.1	-4.91	4.06	0.015
0.2	-6.34	12.15	< 0.001
0.3	2.08	-5.54	0.005
0.4	4.24	-9.86	0.001
0.5	3.60	-10.48	< 0.001

In **CartPole-v1**, every tested σ yields $p < 0.05$: Recurrent-PPO underperforms clean-PPO at low noise ($\sigma \leq 0.3$, $\Delta\bar{r} < 0$) and significantly outperforms it at high noise ($\sigma \geq 0.4$, $\Delta\bar{r} > 0$). In **Highway**, all p-values are also below 0.05: Recurrent-PPO is significantly worse at $\sigma \leq 0.2$ and significantly better at $\sigma \geq 0.3$. Thus, recurrence hurts in clean settings but becomes a clear benefit as observation noise increases.

5 Conclusion

In this study, we compared the robustness of four proximal policy optimization (PPO) variants—feed-forward PPO, Recurrent PPO, Noisy-PPO, and Recurrent-Noisy PPO—under Gaussian observation noise on CartPole-v1 and Highway-env. All agents were trained for 100,000 timesteps over five random seeds, and their performance degradation was quantified by both mean return curves and normalized area-under-degradation-curve (AUDC) metrics. Our results clearly demonstrate that while standard feed-forward PPO suffers a steep decline in performance as the noise level increases, incorporating noise during training or introducing recurrent memory mechanisms can substantially mitigate this brittleness.

Recurrent PPO, which integrates an LSTM layer into the policy network, was able to flatten the degradation curves relative to the feed-forward baseline. However, this improvement comes at the cost of reduced clean-environment performance, as the recurrent model started from a lower mean return when no noise was present. In contrast, Noisy-PPO, which injects Gaussian noise into observations during training, exhibited both high initial returns and exceptional resilience to increasing noise levels, yielding the highest normalized AUDC score in both environments. The combination of recurrence and noise injection in Recurrent-Noisy PPO performed similarly to Noisy-PPO, indicating that memory benefits are marginal when realistic perturbations are already applied during training.

Statistical analysis via paired two-sided t-tests confirmed that the robustness gains achieved by Noisy-PPO and Recurrent-Noisy PPO over the clean feed-forward baseline are significant across all tested noise levels (\$p<0.05\$). These findings underscore that a simple, low-overhead strategy of training-time noise augmentation can deliver robust policies capable of maintaining high performance under substantial observation corruption. While recurrence can aid in partially observable settings, its primary value appears in scenarios where no noise injection is used, and its utility diminishes when combined with noise-augmented training.

Despite these encouraging outcomes, our experiments are limited to two discrete-action benchmarks and a single type of test-time corruption. Future work should extend this analysis to continuous-control tasks and more complex, high-dimensional environments. It will also be valuable to explore a broader set of corruption models—such as sensor dropouts, adversarial perturbations, and varying delay profiles—to assess the generality of noise-based robustness techniques.

5.1 Future Improvements

Building on our findings, several avenues can further enhance the resilience of PPO agents. First, introducing diverse corruption types during evaluation, beyond Gaussian noise, such as action delay, can reveal new failure modes and guide more comprehensive defenses. Second, extending training durations and incorporating curriculum schedules that adaptively increase perturbation intensity may promote more robust feature representations and memory utilization. Third, scaling experiments to include a wider range of environments, spanning continuous, control benchmarks like MuJoCo, high-dimensional vision-based tasks, and real-world simulators—will test the practicality of these methods in complex domains. By systematically diversifying the corruptions, lengthening training regimes, and broadening environmental diversity, future research can deepen our understanding of robust policy learning and accelerate the deployment of reliable reinforcement learning agents in real-world applications.

6 Responsible Research

6.1 Reproducibility and Transparency

Reproducibility is essential for credible machine learning research [4]. To this end, we publish our full training and evaluation pipeline, including ‘train.py‘, ‘evaluate.py‘, and the final version of ‘default.yaml‘, alongside supplementary files. All randomly chosen seeds (42, 123, 587, 1026, 2025) and hyperparameters (learning rate, clip range, LSTM hidden size, noise levels) are fixed and documented in the ‘default.yaml‘ file. We also provide raw CSV outputs of model training and testing used by the provided plotting scripts required to regenerate every figure, ensuring that our results can be independently verified and extended.

6.2 Ethical and Societal Implications

Robust RL agents have dual-use potential: while improved resilience to sensor noise and latency can enhance safety in applications like medical robotics or autonomous transport, the same methods may be co-opted for harmful military or surveillance systems. Schmid et al. [11] analyze how AI innovations diffuse between civilian and defense research, and we urge practitioners to adhere to the EU's *Ethics Guidelines for Trustworthy AI* [5] when deploying robust agents.

Training and evaluating deep RL models also carries an environmental footprint. Although our experiments consume a modest 100 000 timesteps per run, recent studies show that large-scale model training can emit hundreds of kilograms of CO_2 [14]. Therefore, in our early stages of algorithm testing, we used less computing power and ran smaller experiments, increasing it only for the final stages.

Finally, while our benchmarks use synthetic environments without human-sourced data, responsible research demands transparency in data practices. We follow the principle of *Datasheets for Datasets* [3], providing clear documentation of observation wrappers, noise parameters, and evaluation protocols. This level of transparency helps ensure that our conclusions are well-grounded and that downstream users can test the method's applicability to their own domains.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016. URL `https://arxiv.org/abs/1606.01540`.

[2] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://arxiv.org/abs/1706.10295`. Published as a conference paper at ICLR 2018.

[3] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021. doi: 10.1145/3458723.

[4] Benjamin J. Heil, Michael M. Hoffman, Florian Markowetz, Su-In Lee, Casey S. Greene, and Stephanie C. Hicks. Reproducibility standards for machine learning in the life sciences. *Nature Methods*, 18:1132–1135, 2021. doi: 10.1038/s41592-021-01256-7.

[5] High-Level Expert Group on Artificial Intelligence, European Commission. Ethics guidelines for trustworthy AI. European Commission Digital Strategy, April 2019. URL `https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai`. Accessed: 2025-06-22.

[6] Kaixuan Huang and et al. Curriculum-based noise scheduling for robust reinforcement learning, 2021. URL `https://arxiv.org/abs/2108.12345`.

[7] Édouard Leurent. highwayenv: A gym-like environment for autonomous driving. GitHub repository, 2020. `https://github.com/eleurent/highway-env`.

[8] Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4 (1):276–315, 2022. doi: 10.3390/make4010013. URL `https://www.mdpi.com/2504-4990/4/1/13`.

[9] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR, 2017. URL `http://proceedings.mlr.press/v70/pinto17a/pinto17a.pdf`.

[10] Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Generalization, mayhems and limits in recurrent proximal policy optimization, 2022. URL `https://arxiv.org/abs/2205.11104`.

[11] Stefka Schmid, Thea Riebe, and Christian Reuter. Dual-use and trustworthy? a mixed methods analysis of ai diffusion between civilian and defense r & d. *Science and Engineering Ethics*, 28, 2022. doi: 10.1007/s11948-022-00364-7.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: https://www.nature.com/articles/nature16961.

[14] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355.

[15] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017. doi: 10.1109/IROS.2017.8202133. URL `https://arxiv.org/abs/1703.06907`.

[16] Huan Zhang, Hongge Chen, Duane Boning, and Cho Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary, 2021. URL `https://arxiv.org/abs/2101.08452`.

A Usage of LLMs

During the making of the final report and the code related to the project, large language models (LLMs) were used as auxiliary tools to improve both the presentation and technical quality of the work. Specifically:

A.1 Final Report

LLMs were used to refine the structure and wording of sections, ensure consistent use of terminology, and apply appropriate academic style. In particular, the models suggested improvements to LaTeX commands, helped align formatting of equations and references, and provided alternative phrasings to enhance readability and professionalism.

For complex descriptions and definitions—such as those of brittleness, robustness, and corruption mechanisms—LLMs generated concise rewrites that preserved technical precision while improving narrative flow. All reworded passages were carefully reviewed by the authors and, when necessary, further edited to ensure accuracy and coherence.

A.2 Code

In the development of training, evaluation, and especially plotting scripts, LLMs assisted with diagnosing syntax errors, suggesting debugging strategies, and recommending best practices for code organization. They also formatted code snippets for inclusion in the paper and clarified the usage of specific APIs (e.g., Stable Baselines 3, Gymnasium).

While LLMs provided valuable technical and stylistic suggestions, every recommendation was manually verified, tested within the codebase, and adjusted as needed to capture the intended idea.

B Training Execution

Training jobs were run on three platforms, depending on the computational demand and the expected execution time:

Small-scale tests, such as exploring a handful of hyperparameter settings, or plotting were run on a local workstation equipped with an Intel i7 CPU and a single Intel(R) UHD Graphics GPU. This immediate feedback loop enabled quick fixes without waiting for external resources.

When the TU Delft supercomputer queue introduced delays for mid-sized jobs (for example, preliminary noise-injection sweeps or shorter ablation studies), we switched to Google Colab's free GPU instances. Colab Pro was also used for more intense jobs, where GPUs were utilized. Colab provided uninterrupted runtime and sufficient memory to handle our setups, allowing us to continue experimentation, regardless of TU Delft supercomputer.

The final production runs, such as full-scale training of PPO variants across two environments, were executed on the TU Delft HPC cluster. By leveraging multi-GPU partitions and controlled resource allocations, we completed these large jobs efficiently and reproducibly.