Beyond CAGE: Investigating Generalization of Learned Autonomous Network Defense Policies

Melody Wolk, Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, and Adrian Webster
Apple
Cupertino, CA

{ melody_wolk, aapplebaum, cdennler, pnd, mmoskowitz, harold_nguyen, nicole_nichols, n_park, prachwalski, frau, adwebster } @apple.com

Abstract

Advancements in reinforcement learning (RL) have inspired new directions in intelligent automation of network defense. However, many of these advancements have either outpaced their application to network security or have not considered the challenges associated with implementing them in the real-world. To understand these problems, this work evaluates several RL approaches implemented in the second edition of the CAGE Challenge, a public competition to build an autonomous network defender agent in a high-fidelity network simulator. Our approaches all build on the Proximal Policy Optimization (PPO) family of algorithms, and include hierarchical RL, action masking, custom training, and ensemble RL. We find that the ensemble RL technique performs strongest, outperforming our other models and taking second place in the competition. To understand applicability to real environments we evaluate each method's ability to generalize to unseen networks and against an unknown attack strategy. In unseen environments, all of our approaches perform worse, with degradation varied based on the type of environmental change. Against an unknown attacker strategy, we found that our models had reduced overall performance even though the new strategy was less efficient than the ones our models trained on. Together, these results highlight promising research directions for autonomous network defense in the real world.

1 Introduction

Modern network defense is dominated by human processes, such as alert triage and incident response. *Playbook automation* [36, 17, 25] can alleviate human cognitive fatigue by standardizing human-crafted decision logic, but is brittle compared to the fully-automated potential of RL-based network defense agents, as shown in recent RL successes such as Starcraft [39], DotA 2 [5], and Stratego [27]. Using RL agents to perform **fully autonomous** network defense [10, 23, 12, 13, 32] could allow analysts to instead focus their attention on sophisticated scenarios and improved response in general.

This work complements current research by analyzing multiple RL approaches – specifically, variants of Proximal Policy Optimization (PPO) – as applied to the *CAGE Challenge* [2], a public competition to build automated network defender agents, as well as to real-world integration scenarios. The challenge uses a high-fidelity network simulator (*CybORG* [35]) that provides a realistic simulation of an attacker on an enterprise network. Using CybORG, we make the following contributions:

Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

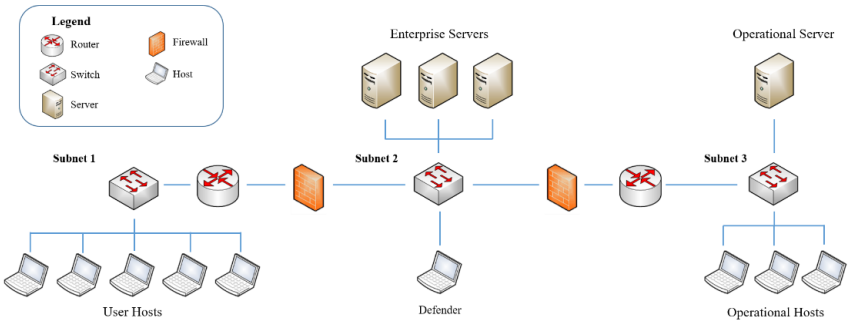

Figure 1: Visualization of CybORG Scenario 2 (taken from [2]). Subnet 1 consists of user hosts; Subnet 2 of important enterprise servers; and Subnet 3 of a critical operational server and three user hosts.

- We describe autonomous network defense methods and performance as tested in the CAGE Challenge, including hierarchical RL, ensemble RL, action masking, and transfer learning.
- We analyze each of our approaches' ability to generalize to unseen network environments and against an unseen but inefficient attacker strategy.

Our results demonstrate that ensemble RL is an effective and promising methodology for building an autonomous network defender. However, they also show that all of our approaches struggle when tested in an unseen environment – being sensitive to host information and attack path – or against an unseen attacker, with results aligning with the stochasticity of the attacker. These contributions both show positive results for new RL methodologies applied to autonomous network defense, as well as identify challenges for real-world integration of such methodologies.

The rest of this paper is organized as follows: Section 2 describes the CAGE Challenge, CybORG, and related work; Section 3 details our approaches' methodology; Section 4 provides initial results and analysis; Section 5 tests our approaches in scenarios that emulate real-world challenges; Section 6 outlines areas for future work; and Section 7 summarizes main points. Additionally, Appendix A includes details on model hyper-parameters, training schedules, and additional metrics.

2 Background

The problem of autonomous network defense is to determine the optimal response given a series of observations – typically *intrusion alerts* – from an enterprise network. These observations contain both false positives – i.e., benign user activity erroneously flagged as malicious – and false negatives, i.e. the alerting components not identifying the attacker, of which the defender needs to balance.

Research to solve this problem includes applying heuristics [6], expert systems [28], model-based reasoning [37], game theory [42], and formal planning [24]. Recent research has also suggested to use RL; early work [32, 9] used highly abstracted network simulators to experiment with multiple RL approaches, including Tabular Q-Learning, Upper Confidence Bound 1 [4], Discounted Upper Confidence Bound [11], and others. Other approaches moved past pure network simulation towards emulation: [23] used containers to emulate a real network, testing RL techniques Ape-X Deep Q-Networks (APEX-DQN) [15], PPO [33], and Asynchronous Advantage Actor-Critic (A3C) [22]. Most of these works used RL techniques without modification, though [12] tests PPO, REINFORCE [30], and a custom auto-regressive PPO, all within an abstract network simulator. In almost all cases, the RL approaches achieved success, bringing the added benefit that these approaches are able to *learn* the dynamics of the defended network without an explicit blueprint.

2.1 CybORG

The CybORG simulator provides an interface for attacker and defender agents to interact with a complex network simulation environment, allowing for detailed encoding of real-world scenarios that is abstracted through a series of wrappers enabling an RL interface while maintaining good realism. Specific environments are encoded as *scenarios*, with the second edition of the CAGE Challenge using *Scenario 2* (Figure 1). Simulations in CybORG are played over these scenarios as *games*

Method	Total Reward	30 Steps		50 Steps		100 Steps	
		BLine	Meander	BLine	Meander	BLine	Meander
PPO Baseline	-95.47	-8.75	-7.31	-14.92	-12.04	-30.19	-22.21
Heuristic Restore	-823.89	-58.33	-44.29	-93.05	-85.75	-184.34	-192.63
Random	-2015.61	-154.06	-33.43	-347.1	-171.49	-726.92	-566.41
Sleep	-3112.2	-218.65	-39.31	-480.17	-267.6	-1134.03	-972.43

Table 1: Baseline results provided by the CAGE Competition organizers [2]. The defender seeks to maximize the total reward, with a best possible total of 0, indicating the attacker has established no footholds and the defender has not taken any Restore actions. Note that we omit the *Sleep* attacker in the interest of space.

between the attacker and defender agent, and also feature a *green agent* that simulates user activity and generates false positives for the defender. Each turn the attacker agent can scan hosts and subnets, launch exploits for lateral movement, escalate privileges, or disrupt a compromised host. In response, the defender can do nothing (*Sleep*); *Monitor* the entire network for malicious activity; *Analyze* a specific host to identify malicious artifacts; *Remove* known malicious artifacts from a host; *Restore* a host to a known good state and remove any attacker foothold; or instantiate one of seven *Decoy* services on the target host. Each turn the defender receives an observation of the current network state as a *bit-vector* – where each host is represented as a fixed set of bits – and a numeric reward. The value of the reward is determined by the severity of the attacker's access to each host, with the defender also receiving a penalty each time it uses the *Restore* action; the reward is always negative such that a game score of 0 indicates a perfect defense.

The CybORG environment allows construction of attacker agents; e.g. [38] used CybORG to show how hierarchical deep RL can build a better attacker agent than DQN. For CAGE, the attacker is restricted to using one of three built-in fixed-approach strategies. The first is the *BLine* attacker agent, which is pre-configured with full knowledge of the network . The second is the *Meander* attacker agent, which needs to discover network details during the game, and seeks to compromise each subnet before moving to the next one. The last is a *Sleep* agent, which executes a no-op each step.

2.2 CAGE

In the CAGE Challenge [2], participants submit defender agents that are ran within the CybORG simulator. For evaluation, each agent is run against the three attackers mentioned above at three different game lengths (30, 50, and 100). The agent's final score is the summation of all rewards received against each attacker at each step length. This construction gives CAGE two primary challenges: (1) identify an optimal policy given noisy network observations, and (2) defend the network without explicitly knowing which attacker strategy is in use. To accompany the challenge, baseline results are provided for a PPO-based agent, a Heuristic agent using only the Restore action, an agent that chooses actions randomly, and a "Sleep" agent. Table 1 reproduces these results, with the PPO agent significantly outperforming the other baselines by all measures.

[10] describes the solution to the winner of the first edition of the CAGE Challenge [1] using Hierarchical PPO (HiPPO). They train two PPO policies, one against BLine and another against Meander. They then train a third policy that seeks only to deploy the pre-trained BLine or Meander policies.

3 Approaches

Each of our approaches build on Proximal Policy Optimization (PPO) [33] as the core RL algorithm. All models were implemented using the RLlib [19] or Stable Baselines3 [31] libraries, which both provide access to PPO out-of-the-box; unless otherwise mentioned, we use RLlib by default. For exploration, models built with RLlib use Curiosity [26] and for Stable Baselines3 use the default Gaussian noise mechanism. Tables 7 and 8 in Appendix A provide additional model information.

Attacker Randomization To avoid the defender from over-fitting to a specific attacker strategy we varied the specific attacker during training. Each time an episode ends and the environment is reset, with probability $Pr(\texttt{BLine})$ we initialize the environment with the BLine attacker, and with probability $Pr(\texttt{Meander}) = 1 - Pr(\texttt{BLine})$ we initialize the environment with the Meander attacker.

Method	Total Reward	30 Steps		50 Steps		100 Steps		CAGE Place
		BLine	Meander	BLine	Meander	BLine	Meander	
Multi-Ensemble	-57.14	-3.44	-6.46	-6.07	-10.22	-11.5	-19.45	2
Ensemble	-58.62	-3.6	-6.55	-6.14	-10.45	-12.08	-19.81	6
Transfer Learning	-69.91	-5.6	-6.43	-9.5	-10.34	-19.82	-18.24	8
TL + Embedding	-72.26	-4.49	-6.31	-8.07	-11.36	-17.72	-24.31	9
Tuned PPO	-73.44	-4.74	-8.06	-8.17	-12.6	-17.04	-22.82	10
HPPO	-120.15	-4.68	-9.16	-8.45	-22.25	-17.72	-57.89	14
Full Visibility	-54.73	-3.9	-5.47	-6.97	-8.27	-14.94	-15.18	-

Table 2: Results for methods in Section 3 after running 1000 episodes against BLine and Meander and with game lengths of 30, 50, and 100; *Total Reward* shows the aggregated result across each step number and attacker agent. Additional information on variance and seed sensitivity can be found in Appendix A.1. The right-most column shows the placement of each model within the CAGE Challenge, with full results available at [2].

This idea factored into many of our approaches discussed later in this section. Additionally, we also trained a single PPO model with Stable Baselines3 and Optuna for hyper-parameter search against an environment with $Pr(\texttt{BLine}) = Pr(\texttt{Meander}) = 0.50$; we refer to this model as ***Tuned PPO***.

Transfer Learning Another approach to combat attacker strategy uncertainty is to use a simple transfer learning procedure where we continually train the same agent on different CybORG configurations, each in sequence. These configurations vary which attacker agent the defender is trained against as well as how many steps each game lasts. Additionally, each iteration features a grid search over several PPO and Curiosity hyper-parameters, with the best performing configuration used to continue training and the others discarded. We refer to this model as ***Transfer Learning***.

Action Masking Action masking is a common technique used to avoid executing known invalid actions. Prior work [16] has shown that even when the mask is removed, the agent is still able to produce useful behaviors. We explored multiple approaches to leverage this idea, ultimately settling on one that used all actions but learned an embedding during training. Additionally, we used a similar training schedule as for Transfer Learning alongside an attacker randomization procedure to avoid overfitting each iteration. We refer to the resulting approach as ***TL + Embedding***.

Hierarchical PPO The idea behind this approach is similar to [12], where an action is decided, followed by a host to perform that action on. Our implementation differs as we use the RLlib multi-agent training environment as opposed to an auto-regressive model: in our approach, we implement a *high-level* policy that identifies what *action* to take, and a *low-level* policy that identifies what *target* to take that action against. To ensure attacker diversity during training, each time an episode ends it is reset with the BLine, Meander, or Sleep attackers, picking each in sequence. We refer to this model as ***HPPO***, differing from the other hierarchical *HiPPO* method [10] mentioned earlier.

Ensemble RL Our last approach uses a simple ensemble technique consisting of individual PPO models trained with different attacker randomization settings and various PPO hyper-parameters. Our ensemble uses weighted majority voting [40], choosing the action preferred by most models and breaking ties by weighing votes by each model's score; these scores are generated beforehand by testing each model against the CAGE Challenge procedure, assigning the result as the model's score. Using this approach we generate two agents for the CAGE Challenge. The first, ***Ensemble***, is one of the higher-performing ensembles that we constructed and consists of seven different PPO models. The second, ***Multi-Ensemble***, uses the same idea but consists of five distinct *ensembles* as opposed to individual models. Appendix A, Table 9 provides more information about each ensemble.

4 Results

We test each of our approaches using a similar process as in the CAGE Challenge itself, aggregating the average total reward over 1000 episodes against BLine and Meander at game lengths of 30, 50, and 100. Data from game lengths 30 and 50 are unique and not derived from intermediate results of 100. Table 2 presents our results, with *Total Reward* showing the result for each defensive approach. Overall, the Ensemble and Multi-Ensemble approaches score best at -58.62 and -57.14 respectively.

Given the best possible reward is 0, Multi-Ensemble appears strongest; while we found a high variance based on seed for these results, anecdotally Multi-Ensemble almost uniformly edges out Ensemble by a point or two. Both approaches perform quite well against BLine, scoring five points better than any non-ensemble approach in the 100 Steps case. Against Meander, we find both of their scores within a few points of our other approaches, in some cases even being behind another.

On the other end of the spectrum, HPPO performs relatively poorly, scoring worse than the provided PPO baseline in Table 1 and the least of our approaches. Interestingly, while this approach performed poor in aggregate, its score against BLine was on par with TL + Embedding and Tuned PPO, and outperformed Transfer Learning. With additional tuning, such as a more exhaustive hyper-parameter search or using an alternative training schedule, this approach might have been stronger.

Transfer Learning, TL + Embedding, and Tuned PPO all performed well, scoring -69.91, -72.26, and -73.44 respectively. Of the three, Transfer Learning had the most notable results, outperforming both Ensemble and Multi-Ensemble for Meander at 30 and 100 Steps. This strong performance is likely due to the high ratio of trials trained against Meander versus BLine; while Transfer Learning was the strongest approach against Meander, it was one of the worst against BLine. TL + Embedding, despite scoring close to Transfer Learning in aggregate, had a very different profile, performing well against BLine but struggling against Meander. Interestingly, even though it scored best among our approaches against Meander at 30 Steps, this model was one of the worst against Meander at 100 Steps. This difference between the two may be due to the training schedule: Transfer Learning trained exclusively on BLine *or* Meander, whereas TL + Embedding used a percentage-wise split.

In the remainder of this section we attempt to better elicit the efficacy of each of our approaches, comparing their performance in the CAGE Challenge as well as to a defender that has full visibility of the attacker's state. We also looked at the action distribution of each approach; this did not reveal any particular insights, but we include it in Appendix A.2 for interested readers. We do note that based on this analysis future work should seek to examine *explainability* for RL-based approaches.

Performance in CAGE Table 2 in the right-most column shows the placement for each of our models within the CAGE Challenge. Multi-Ensemble did best, taking second place to team *CardiffUniv*'s HiPPO implementation, which scored -54.57 for clear first. Third, fourth, and fifth place were from the authors of [10], each using a HiPPO variation and scoring -57.05, -57.29, and -57.46. The seventh place team (*UoA*) had a score of -69.88 and used a Belief Ensemble approach. Based on these results and the competitiveness of the Challenge, we felt that our models performed well.

Full Visibility As another point of comparison, we train a model under conditions with no noise – i.e., no false positives or negatives – and knowledge of the attacker strategy, using PPO-with-Curiosity in RLlib with mostly default hyper-parameters and assuming a top-level model that deploys the correct sub-model. We refer to this model as ***Full Visibility***, with results shown in the last row of Table 2. Compared to this new model, our approaches perform well: Ensemble and Multi-Ensemble are both within a few points of Full Visibility and the others are not far behind; in fact, both ensemble approaches *outperform* Full Visibility against the BLine attacker. This is likely explained by the relative determinism of each attacker strategy: as BLine follows a fixed path, it is easier to anticipate its movements, and therefore the impact of no false positives or negatives is less significant. That our models perform so closely to this agent suggests that they do offer good performance.

5 Real World Integration

Our results strongly suggest that various RL techniques can be used to construct an effective autonomous network defense agent. However, in practice integrating an RL-based defender into a real system is fraught with complexities. For one, training an agent and running live attacks against a production network can significantly impact operations. An alternative is to train on an *emulated* network setup to mimic the production network, but this is not without drawbacks either: [18] trained an automated attacker agent in a fully virtualized network with only a small number of hosts, finding the length of a full episode ranging from 20 to 30 minutes. At one million trials, the required time to train would take on the order of years; *distributed training* can reduce this, but brings with it different costs. An effective alternative is to train the defender within a high-fidelity simulator, as simulations (1) do not impact the production network; (2) can run very quickly; and (3) do not have stringent hardware requirements. However, this approach requires that the simulator maintains an accurate

Model	Total Average Decision Time (ms)	Avg. Decision Time by Attacker (ms) BLine	Meander
Heuristic Restore	0.006 ± 0.001	0.006	0.006
Multi-Ensemble	27.80 ± 3.9	28.82	27.42
Ensemble	6.67 ± 0.38	6.67	6.68
Transfer Learning	0.81 ± 0.07	0.81	0.80
TL + Embedding	0.70 ± 0.1	0.71	0.69
Tuned PPO	0.27 ± 0.05	0.27	0.26
HPPO	1.12 ± 0.47	1.38	1.01

Table 3: Average time to compute a single action in milliseconds, with the total listed with a 95% confidence interval. Averaged over 10000 decisions, consisting of 500 games of length 100 against both BLine and Meander.

representation of the target environment for the RL agent to train in. In this section, we explore integration challenges of a simulation-based approach to (1) make timely decisions; (2) generalize to unseen network environments; and (3) provide robustness against unseen attacker strategies.

5.1 Computation Time

Table 3 lists the average time it took in milliseconds for each model to make a decision, computed over 500 episodes of length 100 against BLine and Meander. The Heuristic Restore baseline is clearly the fastest, taking only 6 *micro*seconds, well below the RL models. This is expected as this approach only looks at the previous observation, and is akin to e.g. responding immediately to a known alert. Within the RL approaches, Tuned PPO is clearly fastest at 0.27ms. TL + Embedding and Transfer Learning both take a similar amount of time at 0.7ms and 0.81ms respectively. We suspect Tuned PPO's speed advantage is due to Stable Baselines3 evaluating more quickly than RLlib. The last three approaches all take longer, with HPPO quickest at 1.12ms, Multi-Ensemble slowest at 27.8ms, and Ensemble between the two at 6.67ms. This distribution aligns with the number of constituent models each of these three has, and is consistent with multiplying the average time for e.g. Transfer Learning with the number of models within each approach (two for HPPO, seven for Ensemble, and 37 for Multi-Ensemble). Looking at the rightmost two columns of the table, we do not see any significant difference in decision time based on attacker for any of the models.

As all of the RL approaches average well under a second, it does not appear that computation time would be a significant challenge for deployment. However, with some changes – such as a larger network or action/observation space, or if Multi-Ensemble grows to use thousands of models – timing may be more impactful; in these cases, distributed evaluation can help speed up the computation.

5.2 Topology Generalization

In this section we analyze how well our approaches perform when evaluated in an environment different than the one they were trained in. This comparison helps serve as a proxy for a deployment scenario where the RL agent is trained in a simulator that is mismatched from the actual network it is supposed to defend. Such a scenario has real-world plausibility: a defender *should* have full knowledge of their environment, and can accurately encode it within a simulator. On the other hand, in practice proper asset management can be challenging for even sophisticated organizations.

We expect differences between training and evaluation environments to have a significant impact on performance due to the observation structure within CybORG: each bit in the observation refers to a specific quality for a specific host, and thus the defender will learn very *explicit* dynamics for each individual host during training. Indeed, this problem was highlighted in [3] which showed that the bit-vector approach did not generalize to unseen environments when using Tabular Q-Learning.

To test this, we introduce three new scenarios that modify the original *Scenario 2* used in the CAGE Challenge, referring to these as *Scenario 3*, *4*, and *5*. Each scenario changes the *OS image* – and corresponding services – for a select set of hosts and/or changes the *attack path* and the order in which the attacker discovers new hosts. Scenarios 3 and 4 are both isomorphic with Scenario 2, swapping names between four *User* hosts for Scenario 3, and swapping names between two *Enterprise* servers for Scenario 4. Both scenarios modify the OS images for these swapped hosts, with Scenario 3 also modifying the next-hop discovery path. Scenario 5 is not isomorphic to Scenario 2, and instead adds

	Model	Total Reward	Percent Change	30 Steps		50 Steps		100 Steps	
				BLine	Meander	BLine	Meander	BLine	Meander
Scenario 3: Image + Path	Multi-Ensemble	-166.2	**-190.8**	-13	-8.5	-22.9	-14.4	-54.5	-27.6
	Ensemble	-147.2	**-154.5**	-13.1	-8.7	-16	-14.5	-56.9	-28
	Transfer Learning	-131.9	**-88.7**	-13.5	-8.1	-23.5	-13.4	-50.6	-22.8
	TL + Embedding	-166.2	**-129.9**	-14.9	-8.5	-27.9	-16.5	-61.5	-36.8
	Tuned PPO	-95.2	**-29.6**	-8.7	-8.3	-14.5	-13	-27.3	-23.4
	HPPO	-176.9	**-47.3**	-11.8	-9.1	-23.2	-22.1	-55.2	-55.6
Scenario 4: Image	Multi-Ensemble	-172.3	**-201.6**	-14.4	-10	-26.3	-20.2	-56.3	-45.1
	Ensemble	-183.6	**-213.2**	-14.9	-10.1	-28.6	-20.6	-63.2	-46.2
	Transfer Learning	-132.4	**-89.4**	-10.7	-9.1	-18.5	-17.3	-38.7	-38.2
	TL + Embedding	-172.5	**-138.7**	-11.6	-9.5	-22.7	-21	-55.9	-51.9
	Tuned PPO	-165.6	**-125.5**	-14.9	-10.7	-27.2	-19.4	-54.1	-39.3
	HPPO	-255.6	**-112.7**	-16.8	-12.9	-31.7	-33.3	-76.3	-84.7
Scenario 5: Path Addition	Multi-Ensemble	-71.2	**-24.5**	-6	-6.5	-9.9	-10.1	-19.4	-19.3
	Ensemble	-74.1	**-26.4**	-5.7	-6.5	-11.1	-10.2	-20.8	-19.5
	Transfer Learning	-87.6	**-25.3**	-8.5	-6.3	-14.5	-10	-29.8	-18.3
	TL + Embedding	-103.5	**-43.2**	-8.8	-6.3	-16.5	-11.6	-34.7	-25.5
	Tuned PPO	-74.3	**-1.2**	-5	-8.1	-8.5	-13	-16.7	-23.1
	HPPO	-117.4	**+2.3**	-4.5	-8.8	-7.9	-22.2	-16.8	-57.2

Table 4: Results from running each of our six submissions to the CAGE Challenge against Scenarios 3, 4, and 5 from Section 5.2 for 500 episodes of game lengths 30, 50, and 100, including the summed total reward and the percentage change for each agent from their performance in Scenario 2. Percent changes in **bold** highlight learning agents with the *least* impacted performance.

more attack path options for the attacker: namely, in Scenario 2 two of the *User* hosts discover the *Enterprise0* host, and another two *User* hosts discover the *Enterprise1* host. In Scenario 5, we expand this so that all four *User* hosts discover both of the enterprise hosts.

Table 4 provides the results from running each of our approaches against all three scenarios; we tested Heuristic Restore and Sleep and found each of them to score within a few percentage points of their original Scenario 2 score. Additionally, we do not train any RL-based defenders on these scenarios, as due to the similarities with Scenario 2 we would expect results to be the same. The results for our approaches are all almost uniformly worse in each of the three new scenarios. While we anticipated a drop, the magnitude of the difference is significant, particularly for Scenarios 3 and 4 where both ensemble approaches dropped by nearly 200%. The only approach that did not have a huge drop in these two scenarios was Tuned PPO, which only fell 29.6% in Scenario 3. We credit this to Tuned PPO's Gaussian noise exploration strategy, which in some cases leads to local optima [41]; here it may be that Tuned PPO fell into such an optima which was more resilient to the observation space mismatch introduced in this scenario. The only other model that stands out is Transfer Learning, which drops 88.7% and 89.4% in Scenarios 3 and 4. While this is a marked drop in performance, it is the highest scorer for Scenario 4, and is second in score only to Tuned PPO for Scenario 3.

For both Scenarios 3 and 4 the performance against BLine is worse than in the original Scenario 2, likely as a result of BLine taking advantage of the observation mismatch and moving more quickly. For Scenario 3, the Meander scores are similar to Scenario 2 due to the location of the observation changes – Meander will compromise all *User* hosts before moving to the *Enterprise* hosts, by the time of which the observation mismatch is not as relevant. For Scenario 4, however, performance against Meander is notably less than the others, even though this Scenario only modified OS images for two servers and left the attack path unchanged. This drop is likely due to the learned decoys no longer applying; since decoys are OS specific the swap renders the learned decoy strategy ineffective.

Scenario 5 does not prove as challenging for our approaches, with HPPO and Tuned PPO scoring roughly the same as Scenario 2, and the other four models only performing slightly worse. This drop appears to come entirely from BLine; the additional attack paths from the *User* hosts to the *Enterprise* ones does not impact Meander as it must compromise all *User* hosts regardless, and so performance is the same as it was in Scenario 2. By contrast, the BLine attacker will randomly take advantage of the new path, and so we can see performance against it in Scenario 5 slightly worse than Scenario 2. This is most likely explained by each defensive agent effectively learning optimizations to combat the BLine strategy, with this strategy less effective when BLine is given more options.

Model	Meander	RandomMeander	Percent Change	Avg. Decision Time (ms)
Heuristic Restore	-192.62	-175.22	**+10.6**	0.006
Random	-566.41	-428.16	**+24.4**	-
Sleep	-972.43	-794.0	**+18.3**	-
Multi-Ensemble	-19.45	-26.83	**-37.9**	27.44
Ensemble	-19.81	-26.34	**-33**	6.65
Transfer Learning	-18.24	-24.68	**-35.3**	0.81
TL + Embedding	-24.31	-26.95	**-10.9**	0.71
Tuned PPO	-22.82	-37.54	**-64.5**	0.26
HPPO	-57.89	-42.08	**+27.3**	0.97

Table 5: Average total reward and individual decision time for each approach against RandomMeander over 1000 episodes of length 100. Percentage Change shows the relative increase or decrease in reward against RandomMeander as compared to Meander; a positive change indicates better defender performance against RandomMeander, and a negative change indicates worse performance.

We note that Scenarios 3 and 4 are least likely to occur in the real-world as the defender will usually know some facts about each host. Scenario 5 by contrast is much more likely: each host has the same OS image, but the defender only knows a subset of all attack paths in the network. Extrapolating towards a future integration, we would expect to see differences between simulation and deployment be more similar to Scenario 5 than Scenarios 3 and 4.

5.3 Attacker Strategy Robustness

Our last integration analysis looks at the robustness of each approach against an unseen attacker strategy. This is inspired in part by [23] and [12], which both found that RL-based defenders performed well against static attackers, with degraded performance against an RL-based attacker. Our new attacker does not use RL, but instead more simply modifies Meander to allow for *duplicate compromise*. During a game, Meander keeps a list of which hosts it has exploited and is prohibited – regardless of foothold – from exploiting any host it has previously exploited, except in cases where the most recent exploit has failed. This restriction optimizes Meander to avoid exploiting already-compromised hosts, but comes at a cost where it will not re-compromise (select) lost footholds. We create a new strategy – *RandomMeander* – that removes this restriction, allowing the attacker to exploit a random (known) host, regardless if that host has been compromised by the attacker.

Table 5 contains the results from running three of the baseline approaches in Table 1 and our new approaches against RandomMeander for 1000 episodes of length 100. The first three rows of Table 5 show performance against three of the baselines. In each case, the baseline defender performs *better* against RandomMeander than it does against Meander, due in part to the former moving more slowly because it wastes turns by exploiting currently compromised hosts.

By contrast, nearly all of our models perform *worse* against RandomMeander than they do against Meander. Multi-Ensemble, Ensemble, and Transfer Learning all drop over 30%, with TL + Embedding also performing worse, but only 10.9%. These are each large drops, but they only amount to a small number of points (~7) and their scores still remain convincingly higher than the baselines. Interestingly, despite these models varying in performance against Meander, their scores against RandomMeander are within ~2 points of one another. Tuned PPO has an even bigger performance drop, scoring over 60% worse against RandomMeander as opposed to Meander. HPPO differs in that it performs better against RandomMeander: its tiered approach may offer more robustness against attacker variation as it is not as optimized against each attacker strategy's nuances. We do note that the timing against RandomMeander is similar to the timing for BLine and Meander as per Table 3.

Part of RandomMeander's relative success against our defenders is likely explained by it following different paths than Meander: RandomMeander will try to *re-compromise* lost footholds, a strategy our agents have not seen. Looking at trends across agents, we also see that our agents perform better against more deterministic agents – against the baselines, BLine, then Meander, then RandomMeander are best, but against our approaches, RandomMeander, Meander, then BLine are best. We note that even though the reward difference between Meander and RandomMeander is small, it may imply the existence of an untested attacker strategy that is even more effective against our defenders.

6 Discussion and Future Work

We believe that ensemble models could be quite powerful for future autonomous network defense solutions, with our ensembles outperforming other approaches in this work likely due to learning a better decoy strategy (Appendix A.2). For the future, alternative ensemble voting techniques – e.g., Boltzmann multiplication [40] or other heuristics based on constituent model properties – could provide better robustness or performance. Another approach is to include stronger individual models in the ensemble, or using non-PPO models within an ensemble to offer better model diversity and robustness. Additionally, leveraging the results of [10] and using an ensemble of HiPPO model or building a HiPPO model that uses ensembles as sub-policies could be promising.

The larger future challenges stem from safely integrating autonomous RL network defense agents with complex live data and high-consequence security playbooks. While there are numerous sub-challenges from both the environment and integration, we highlight: environment generalization, attacker-defender robustness, human-machine teaming, and insightful measures of system effectiveness.

Generalization to unseen environments is sparsely tested in the literature, with many if not all of the approaches previously mentioned [42, 24, 23, 10, 12] likely to suffer performance losses similar to those tested here. Recent work in [3] modifies CyberBattleSim [21] to add a defender interface and allow for reasoning about host features instead of individual hosts, testing different state space designs. They find that the bit-vector approach used in this work struggles to generalize to unseen environments, but approaches that use more abstract state spaces – e.g., binning the percentage of compromised hosts – offer much better performance in unseen environments. Another recent approach [7] suggests modifying the observation space to use graph-based features to augment the traditional bit-vector representation and support better generalization.

Robustness against attacker strategies also remains an open problem, noted in other autonomous network defense research [23, 12], the latter of which uses *self-play* [34] and *opponent pools* [5] but still struggles to converge on an optimal policy against a dynamic attacker. The authors of [12] show in [13] that by re-framing the network defense problem as one of optimal stopping [8] they can use a game-theoretic solution that is more robust against dynamic attackers. Regardless, future approaches using RL will need to consider robustness against attacker strategy given (1) the adversarial nature of security domain – which puts all system components, including the RL agent itself, in scope – and (2) the potential susceptibility of RL deployments to adversarial examples and other attacks [20].

Lastly, we note that any deployed RL-based defensive agent should be measured against how effective it is in relation to *human operators*. Prior work has looked at proving optimal solutions [24, 13] or automated-only comparative analysis [23], though [29] recently compared automated defender and human performances, and [14] ran an end-user study showing usability of an automated attacker.

7 Conclusion

In this work we examined several RL techniques for autonomous network defense within a high-fidelity network security simulator, as defined in the second edition of the CAGE Challenge. Of our approaches, leveraging an ensemble of PPO-based policies and using a majority voting scheme was most effective. This strategy outperformed our other approaches, ranked second place in the CAGE competition, and scored similarly to an RL-based defender with full network visibility. We believe ensemble approaches will push the state of the art for autonomous network defense agents.

To understand the practical viability of RL-based approaches, we also explored challenges to real-world integration for autonomous network defense, including timing, generalization to unseen environments, and robustness to attacker strategy. Timing did not appear to be a factor for this set of methods, with all models performing within reasonable bounds. Though the unseen environments differed only slightly from the training environment, all of our approaches performed significantly worse, with the ensemble approaches degraded the most; we observed that modifying hosts caused the most disruption to our defenders, with adding attack paths not being as impactful. For robustness against alternative attacker strategies, we discovered that testing against an unseen but *less* efficient attacker in fact was *harder* for our approaches to defend against, likely due to the unseen attacker being more unpredictable and executing attack paths previously unseen. Combining these conclusions, though RL approaches can create effective defenders in some limited scenarios, integration to a production system will require significant future work and investment.

References

[1] Cyber autonomy gym for experimentation challenge 1. https://github.com/cage-challenge/cage-challenge-1, 2021. Created by Maxwell Standen, David Bowman, Son Hoang, Toby Richer, Martin Lucas, Richard Van Tassel.

[2] Cyber autonomy gym for experimentation challenge 2. https://github.com/cage-challenge/cage-challenge-2, 2022. Created by Maxwell Standen, David Bowman, Son Hoang, Toby Richer, Martin Lucas, Richard Van Tassel, Phillip Vu, Mitchell Kiely.

[3] Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, Adrian Webster, et al. Bridging automated to autonomous cyber defense: Foundational analysis of tabular q-learning. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*, pages 149–159, 2022.

[4] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[5] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[6] Curtis Carver, JM Hill, John R Surdu, and Udo W Pooch. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY*, pages 110–116, 2000.

[7] Josh Collyer, Alex Andrew, and Duncan Hodges. Acd-g: Enhancing autonomous cyber defense agent generalization through graph embedded network representation. ICML Workshop on Machine Learning for Cybersecurity, 2022.

[8] Evgenii Borisovich Dynkin. A game-theoretic version of an optimal stopping problem. In *Doklady Akademii Nauk*, volume 185, pages 16–19. Russian Academy of Sciences, 1969.

[9] Richard Elderman, Leon JJ Pater, Albert S Thie, Madalina M Drugan, and Marco A Wiering. Adversarial reinforcement learning in a cyber security simulation. In *ICAART (2)*, pages 559–566, 2017.

[10] Myles Foley, Chris Hicks, Kate Highnam, and Vasilios Mavroudis. Autonomous network defence using reinforcement learning. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 1252–1254, 2022.

[11] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*, 2008.

[12] Kim Hammar and Rolf Stadler. Finding effective security strategies through reinforcement learning and self-play. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[13] Kim Hammar and Rolf Stadler. Learning security strategies through game play and optimal stopping. *arXiv preprint arXiv:2205.14694*, 2022.

[14] Hannes Holm. Lore a red team emulation tool. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[15] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[16] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. In *The International FLAIRS Conference Proceedings*, volume 35, 2022.

[17] IACD. IACD Playbooks and Workflows. https://www.iacdautomate.org/intro-to-playbooks-and-workflows, 2020. [Accessed 2022].

[18] Li Li, Raed Fayad, and Adrian Taylor. Cygil: A cyber gym for training autonomous agents over emulated network systems. *arXiv preprint arXiv:2109.03331*, 2021.

[19] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

[20] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3756–3762, 2017.

[21] Microsoft Defender Research Team. CyberBattleSim. *Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei*, 2021.

[22] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[23] Andres Molina-Markham, Cory Miniter, Becky Powell, and Ahmad Ridley. Network environment design for autonomous cyberdefense. *arXiv preprint arXiv:2103.07583*, 2021.

[24] Scott Musman, Lashon Booker, Andy Applebaum, and Brian Edmonds. Steps toward a principled approach to automating cyber responses. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 110061E. International Society for Optics and Photonics, 2019.

[25] OASIS. OASIS Collaborative Automated Course of Action Operations (CACAO) for Cyber Security TC. "`https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cacao`", 2021. Accessed 2022.

[26] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[27] Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *arXiv preprint arXiv:2206.15378*, 2022.

[28] Phillip A Porras and Peter G Neumann. Emerald: Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th national information systems security conference*, volume 3, pages 353–365, 1997.

[29] Baptiste Prebot, Yinuo Du, Xiaoli Xi, and Cleotilde Gonzalez. Cognitive models of dynamic decisions in autonomous intelligent cyber defense. 2022.

[30] R. J. Williams. Reinforcement-learning connectionist systems, 1987.

[31] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[32] Ahmad Ridley. Machine learning for autonomous cyber defense. *The Next Wave*, 22(1):7–14, 2018.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[35] Maxwell Standen, Martin Lucas, David Bowman, Toby J Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents. 2021.

[36] Rock Stevens, Daniel Votipka, Josiah Dykstra, Fernando Tomlinson, Erin Quartararo, Colin Ahern, and Michelle L Mazurek. How ready is your ready? assessing the usability of incident response playbook frameworks. In *CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2022.

[37] Thomas Toth and Christopher Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, pages 301–310. IEEE, 2002.

[38] Khuong Tran, Ashlesha Akella, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, and Chin-Teng Lin. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv preprint arXiv:2109.06449*, 2021.

[39] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[40] Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

[41] Junwei Zhang, Zhenghao Zhang, Shuai Han, and Shuai Lü. Proximal policy optimization via enhanced exploration efficiency. *Information Sciences*, 2022.

[42] Saman A Zonouz, Himanshu Khurana, William H Sanders, and Timothy M Yardley. Rre: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):395–406, 2013.

A Additional Details

This section describes each approach's specific parameters and training setup; missing parameters should be assumed to align with the RLlib or Stable Baselines3 default. We also include a visualization and discussion of the spread for each model's results from Section 4 – including comments about seed sensitivity – as well as analysis of each model's action distribution.

A.1 Variance and Seed Sensitivity

Figure 2 shows the average total rewards from Table 2 alongside their standard deviations as well as the 25th to 75th percentiles. This visualization highlights the high degree of overlap in each models' results – aside from the results for HPPO against Meander, many of the models' average total reward was within range of the others. We found this overlap to remain even when increasing the number of trials: running several of the models at 5000 trails as opposed to 1000 resulted in the same variance.

We ultimately credit the spread to *seed sensitivty* within CybORG itself: the seed impacts the specific attack mechanics that the red agent uses, which in turn impacts each decoy's efficacy. By contrast, the organizers of the CAGE Challenge fix the seed to a constant value during evaluation, with each of our approaches having a much smaller variance; i.e. Multi-Ensemble had the smallest with a 95% confidence interval of ± 0.59 and HPPO had the largest at ± 1.63. For future work we plan to better analyze the variance and the approach's sensitivity to the seed value, although anecdotally we found that the relative performance of each model stayed consistent across seeds.

A.2 Action Distribution

We sought to analyze the number of "correct" actions each model chose during a game. Lacking an optimal policy to compare to, we instead identified four actions and context where it is clear the action is clearly *wrong* or at least *unhelpful*:

- *Wrong Restores.* The defender restores a host that is not compromised, incurring a penalty.

Approach	Wrong Restores	Wrong Removes	Bad Host Targeting	Sleep
Multi-Ensemble	0.23	1.4	4.72	0.7
Ensemble	0.49	2.46	4.16	1.18
Transfer Learning	0.28	6.27	3.66	0
TL + Embedding	0.33	6	0.91	0
Tuned PPO	0.6	1.81	11.04	1.19
HPPO	0.44	0	0	0

Table 6: Average number of incorrect actions per game by methodology, averaged over 500 games of length 100 against each of BLine and Meander.

- *Wrong Removes.* The defender attempts to remove malware from a non-compromised host.
- *Bad Host Targeting.* The defender acts on a host the attacker is *unable* to scan/interact with.
- *Sleep.* The defender executes a no-op.

Table 6 shows the average number of each category for each of our approaches, averaged over 500 games of length 100 against each of BLine and Meander. Despite each approach having highly varied overall performances from each other, we see that each averages roughly the same number of incorrect Restore actions. We suspect this is due to the penalty for the Restore action, where each model learns more explicitly to avoid incorrect applications of this action.

The other three categories have varied spreads for each approach. Both Ensemble approaches have similar average number of incorrect Remove and Sleep actions, the former of which is to be expected due to noise and the latter from particular evaluation settings. Both Ensembles average ~4 actions that target the unreachable hosts, which we were surprised to see given the attacker always ignores these. Tuned PPO has a similar profile, with a low number for incorrect Remove and Sleep actions, but in this case a much higher number of actions targeting the unreachable hosts. The two Transfer Learning approaches notably have a higher average number of incorrect Remove actions, but do not use Sleep and have fewer actions targeting the unreachable hosts.

Ultimately we did not feel this analysis proved helpful as a proxy for overall performance. Looking at HPPO we see no incorrect Removes, targeting unreachable hosts, or Sleep, but it has the worst overall score among our approaches. Instead, the success of each approach seems more tightly coupled with the specific action distribution used by each approach, visualized in Figure 3. Looking at the figure shows that HPPO overprioritizes the *Monitor* action, while the other five approaches all favor using decoys. The nuance and difference however between these approaches seems to fall more on the side of decoy deployment strategy, with each approach preferring different decoys to use. We suspect that the particular placement of these decoys is an important factor as well, and believe that future work trying to operationalize these ideas should investigate better explaining particular model decisions.

Model	Iteration	$Pr(\texttt{BLine})$	$Pr(\texttt{Meander})$	Game Length	Steps
Transfer Learning	1	0	1	30	400000
	2	1	0	30	400000
	3	0	1	50	400000
	4	1	0	50	400000
	5	0	1	100	800000
	6	1	0	100	800000
	7	0	1	100	400000
TL + Embedding	1	0.95	0.05	30	500000
	2	0.05	0.95	30	500000
	3	0.95	0.05	50	500000
	4	0.05	0.95	50	500000
	5	0.95	0.05	100	800000
	6	0.05	0.95	100	800000
	7	0	1	100	400000

Table 7: Training schedules for the Transfer Learning and TL + Embedding.

Model	Parameter	Value	Parameter	Value
Tuned PPO	activation_fn clip_range gae_lambda learning_rate number_epochs vf_coeff	relu 0.1 0.99 $5.0148 \cdot 10^{-5}$ 10 0.102	batch_size entropy_coef gamma max_grad_norm number_steps	16 0.0002 0.99 0.5 2048
Full Visibility: Meander	entropy_coeff	0.0001	vf_clip_param	100
Full Visibility: BLine	gamma	0.95	entropy_coeff	0.0001

	Parameter	Value
Transfer Learning	entropy_coeff_schedule exploration_config.feature_dim exploration_config.forward_net_hiddens exploration_config.lr lr kl_coeff model.fcnet_hiddens	[[0, 0.001], [1000.0, 0.0001], [10000.0, 10^{-5}], [100000.0, 10^{-6}]] 512 [512] 10^{-4} 9.10^{-5} 0.3 [128, 128]
TL + Embedding	entropy_coeff_schedule exploration_config.feature_dim exploration_config.forward_net_hiddens exploration_config.lr lr kl_coeff model.fcnet_hiddens	[[0, 0.001], [1000.0, 0.0001], [10000.0, 10^{-5}], [100000.0, 10^{-6}]] 512 [512] 10^{-4} 9.10^{-5} 0.3 [128, 128]
HPPO	entropy_coeff_schedule exploration_config.feature_dim exploration_config.lr lr model.policies.high_level_policy.gamma model.policies.low_level_policy.gamma	[[0, 0.001], [1000.0, 0.0001], [10000.0, 10^{-5}], [100000.0, 10^{-6}]] 128 0.001 9.10^{-6} 0.95 0.99

Table 8: Hyper-parameter values for various models.

Ensemble ID + Score	Model	Score	Strategy	Training Steps	Pr(BLine)	Pr(Meander)	gamma	entropy_coeff	model.fcnet_hiddens
1: -59.37	0567a5ffcd1e458fba7bdfa385f299c3	-79.98	Normal	3.0mm	0.25	0.75	0.95	0.0001	[256, 256]
	3aaa5ff3de5a4b19bac0861e83982e91	-72.66	Normal	1.6mm	0.5	0.5	0.96	0	[256, 256]
	541b39af6d0d477cb6b535fa9356e3a0	-77.26	Maintenance	1.6mm	0.5	0.5	0.95	0	[256, 256]
	bc1b3cdd5a704f01be7c6f8847cded57	-78.96	Normal	3.0mm	0.5	0.5	0.925	0	[256, 256]
	c1dc05deffde4d3b9ab9af338844f756	-73.06	Normal	3.0mm	0.5	0.5	0.95	0	[256, 256]
	d03ff193281e4043862c0232779d3e58	-81.46	Maintenance	1.6mm	0.5	0.5	0.95	0.0001	[256, 256]
	ddc46c43b85e4ffc87e18ac6986c1850	-77.47	Normal	3.0mm	0.75	0.25	0.95	0.001	[256, 256]
2: -58.69	3aaa5ff3de5a4b19bac0861e83982e91	-72.66	Normal	1.6mm	0.5	0.5	0.96	0	[256, 256]
	541b39af6d0d477cb6b535fa9356e3a0	-77.26	Maintenance	1.6mm	0.5	0.5	0.95	0	[256, 256]
	b296b7eb79a240e3bf47fd4742146682	-74.29	Normal	3.0mm	0.75	0.25	0.95	0.0001	[256, 256]
	bc1b3cdd5a704f01be7c6f8847cded57	-78.96	Normal	3.0mm	0.5	0.5	0.925	0	[256, 256]
	c1dc05deffde4d3b9ab9af338844f756	-73.06	Normal	3.0mm	0.5	0.5	0.95	0	[256, 256]
	d03ff193281e4043862c0232779d3e58	-81.46	Maintenance	1.6mm	0.5	0.5	0.95	0.0001	[256, 256]
	ed7011ae66ae4f5fac28b384f0c4cb00	-76.57	Normal	1.6mm	0.5	0.5	0.95	0.0001	[256, 256]
3: -58.76	0567a5ffcd1e458fba7bdfa385f299c3	-79.98	Normal	3.0mm	0.25	0.75	0.95	0.0001	[256, 256]
	3aaa5ff3de5a4b19bac0861e83982e91	-72.66	Normal	1.6mm	0.5	0.5	0.96	0	[256, 256]
	541b39af6d0d477cb6b535fa9356e3a0	-77.26	Maintenance	1.6mm	0.5	0.5	0.95	0	[256, 256]
	c1dc05deffde4d3b9ab9af338844f756	-73.06	Normal	3.0mm	0.5	0.5	0.95	0	[256, 256]
	bc1b3cdd5a704f01be7c6f8847cded57	-78.96	Normal	3.0mm	0.5	0.5	0.925	0	[256, 256]
	d03ff193281e4043862c0232779d3e58	-81.46	Maintenance	1.6mm	0.5	0.5	0.95	0.0001	[256, 256]
	ddc46c43b85e4ffc87e18ac6986c1850	-77.47	Normal	3.0mm	0.75	0.25	0.95	0.001	[256, 256]
4: -60.15	3aaa5ff3de5a4b19bac0861e83982e91	-72.66	Normal	1.6mm	0.5	0.5	0.96	0	[256, 256]
	5116aaeeb8234c0980fa0ef4fa409cf0	-117.53	Normal	3.0mm	0.95	0.05	0.9	0	[256, 256]
	541b39af6d0d477cb6b535fa9356e3a0	-77.26	Maintenance	1.6mm	0.5	0.5	0.95	0	[256, 256]
	62f271a5b26a41e2801bd0ed5316f98b	-474.3	Normal	1.5mm	1	0	0.95	0	[128, 128]
	b296b7eb79a240e3bf47fd4742146682	-74.29	Normal	3.0mm	0.75	0.25	0.95	0.0001	[256, 256]
	c1dc05deffde4d3b9ab9af338844f756	-73.06	Normal	3.0mm	0.5	0.5	0.95	0	[256, 256]
	cdf3076e074c4c33b6c538496bd46ed7	-125.68	Normal	3.0mm	0.95	0.05	0.95	0	[256, 256]
	ed7011ae66ae4f5fac28b384f0c4cb00	-76.57	Normal	1.6mm	0.5	0.5	0.95	0.0001	[256, 256]
5: -59.37	0567a5ffcd1e458fba7bdfa385f299c3	-79.98	Normal	3.0mm	0.25	0.75	0.95	0.0001	[256, 256]
	157bde6d96e24388b6d1fe6e6e487e85	-109.25	Normal	3.0mm	0.75	0.25	0.9	0.0001	[256, 256]
	3aaa5ff3de5a4b19bac0861e83982e91	-72.66	Normal	1.6mm	0.5	0.5	0.96	0	[256, 256]
	5116aaeeb8234c0980fa0ef4fa409cf0	-117.53	Normal	3.0mm	0.95	0.05	0.9	0	[256, 256]
	b5c4ca39a6ba4ebab3b1c4c7939427bd	-78.66	Normal	3.0mm	0.75	0.25	0.95	0	[256, 256]
	c1dc05deffde4d3b9ab9af338844f756	-73.06	Normal	3.0mm	0.5	0.5	0.95	0	[256, 256]
	ddc46c43b85e4ffc87e18ac6986c1850	-77.47	Normal	3.0mm	0.75	0.25	0.95	0.001	[256, 256]
	ed7011ae66ae4f5fac28b384f0c4cb00	-76.57	Normal	1.6mm	0.5	0.5	0.95	0.0001	[256, 256]

Table 9: Details for each of the five ensembles. Each ensemble is listed along with its score – used by the Multi-Ensemble method – and its constituent models. In addition to the hyper-parameter listed in the table, all monitors also use the following: `framework` = `torch`, `vf_clip_param` = 100, and `train_batch_size` = 1000. Note that all models were trained with RLlib. The *Maintenance* strategy refers to a strategy where the model was trained specifically to keep the intermediate reward greater than -10.

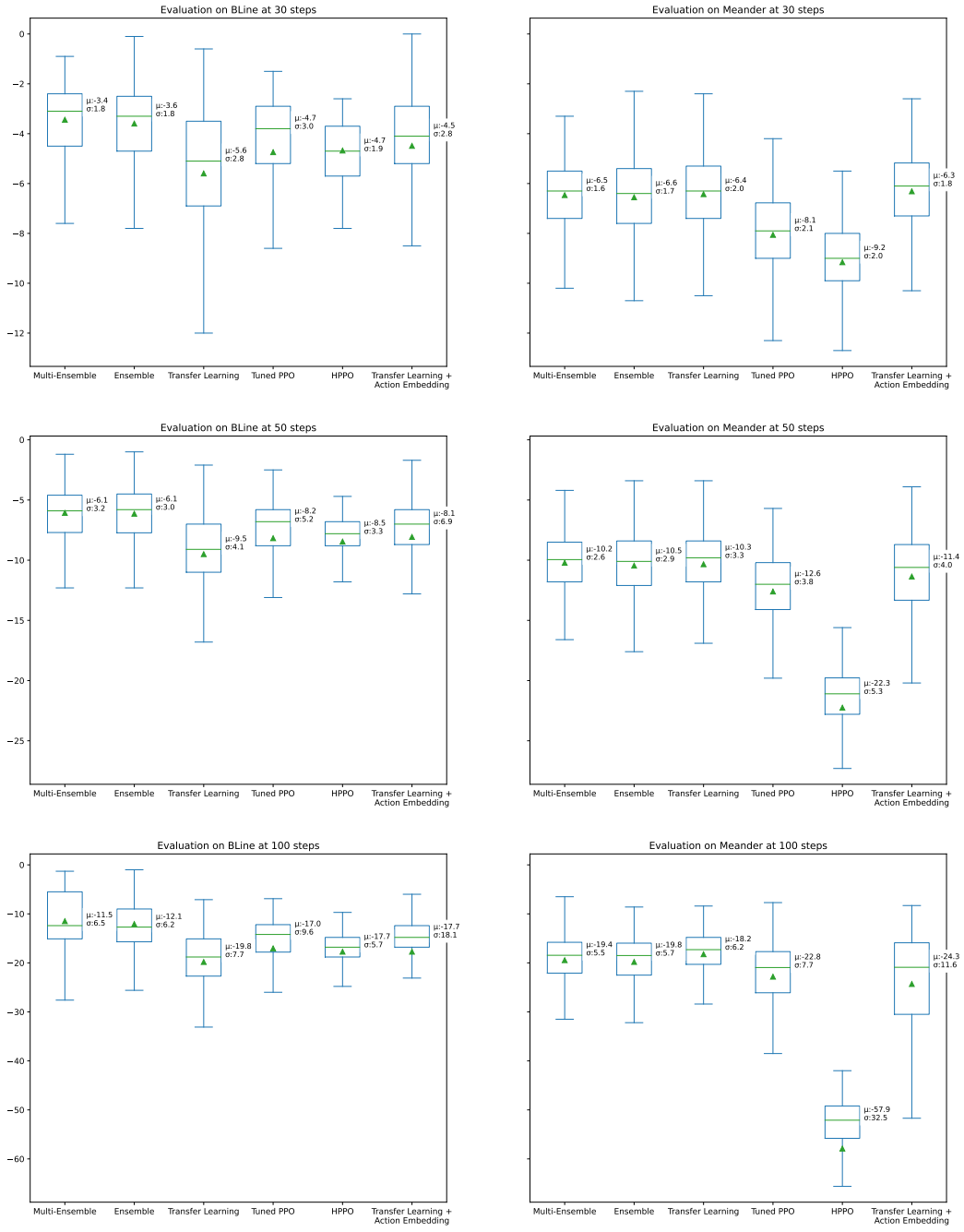

Figure 2: Boxplots showing spread for rewards from select models tested in Table 2. The box spans the range of the 25th percentile to 75th percentile, with the green line showing the median. Additionally, the green triangle shows the average, with the side annotation of μ showing the mean and σ the standard deviation.

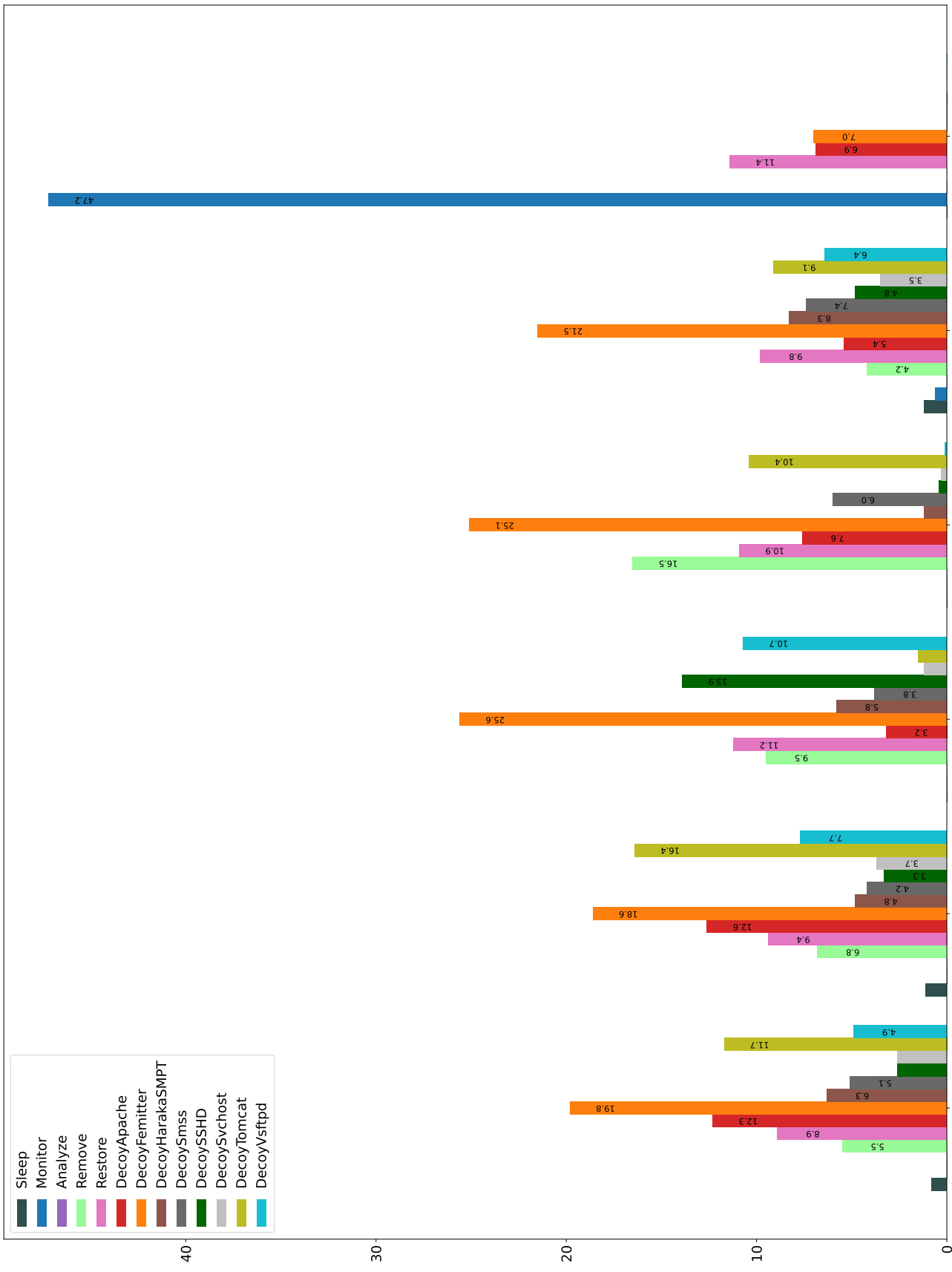

Figure 3: Bar chart showing the average number of actions per action type for each of the six models submitted to the CAGE Challenge. Numbers averaged over 500 episodes, combining results for both BLine and Meander at game length of 100.