GENERALIZABLE GRAPH-BASED REINFORCEMENT LEARNING AGENTS FOR AUTOMATED CYBER DEFENSE

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep reinforcement learning (RL) is emerging as a viable strategy for automated cyber defense (ACD). The traditional RL approach represents networks as a list of computers in various states of safety or threat. Unfortunately, these models are forced to overfit to specific network topologies, rendering them ineffective when faced with even small environmental perturbations. In this work, we frame ACD as a two-player context-based partially observable Markov decision problem with observations represented as attributed graphs. This approach allows our agents to reason through the lens of relational inductive bias. Agents learn how to reason about hosts interacting with other system entities in a more general manner, and their actions are understood as edits to the graph representing the environment. By introducing this bias, we will show that our agents can better reason about the states of networks and zero-shot adapt to new ones. We show that this approach outperforms the state-of-the-art by a wide margin, and makes our agents capable of defending never-before-seen networks against a wide range of adversaries in a variety of complex, and multi-agent environments.

1 INTRODUCTION

Automated cyber defense (ACD) systems are agents which, with no human intervention, defend a network from complex cyber-attacks–automated, or human (Vyas et al., 2023). Like an autonomous security operations center, ACD agents monitor the network at all times, waiting to respond to a cyber-attack. When an incident occurs, the ACD agent would have the power to update firewall rules, reset machines, etc., to prevent the intruder from spreading. Using a simulated network environment, we can train agents for this task using reinforcement learning (RL). Prior works in this field compress the environment into a vector and feed it into deep RL networks (Chai et al., 2020; Chowdhary et al., 2021; Eghtesad et al., 2020; Piplai et al., 2022; Ridley, 2018). This naive approach produces agents that are adept at defending the network they were trained on, but overly sensitive to minor changes in the network topology. As such, these agents are rarely evaluated in new and/or modified environments, a situation that is essential for any technology to be viable in the real world.

This observation coincides with a very old problem in RL: environmental overfitting (Whiteson et al., 2011). In traditional supervised learning, a model has become overfit if it memorizes every data point and is unable to generalize about data it has never seen. To account for this, one would partition the data into disjoint training and testing sets to ensure the model can adapt to uncertainty. In RL, the uncertainty lies in how an action will affect the environment. Environmental overfitting occurs when an agent no longer has uncertainty about the world it interacts with. Admittedly, model overfitting is not a problem in some cases; most RL benchmarks train and evaluate on the same environments (Kirk et al., 2023), and why shouldn't they? An agent that plays chess can always expect the game to start with the same two sets of pieces placed in the same way across the same sixty-four squares. There is no reason to expect a chess-playing agent to adapt to a 9×9 board with five knights. But, for many real-world systems, such as ACD, the assumption that the environment will never change does not hold (Almasan et al., 2022).

One way to represent differences between networks' topologies and encourage agents to learn policies that generalize across environments is to represent them as graphs. The use of graphs for

state representation has been done by several prior works in this field (Ridley, 2018; Collyer et al., 2022; Gangupantulu et al., 2021), but none process the graph directly. Agents in prior work receive graphs representing the environment as part of their observations, but when processing them for their agents, the graph is compressed into a vector, and information is lost. Most often, the topological structure of the graph is thrown out entirely, and information about individual nodes is concatenated together (Piplai et al., 2022; Ridley, 2018; Booker & Musman, 2020; Foley et al., 2022; Walter et al., 2021; Wolk et al., 2022). This results in models that learn about implicit relationships between nodes but cannot adapt to explicit changes of novel network topologies because these changes cannot be communicated to the agents. Perturbations as simple as changing the order in which nodes are indexed can cause these models to perform no better than random.

In this work, we propose a novel, graph-centric strategy for highly generalizable automated network defense agents. Harnessing the power of relational inductive bias (Battaglia et al., 2018), we provide our models with the full graph of the network, which they process without compression using a graph neural network (GNN) (Kipf & Welling, 2016). Inductive GNNs process graph input in a permutation-invariant manner and are structurally unable to rely on fixed node identities or positions. Automorphisms between nodes are implicitly understood to GNN-based models as features and edges are identical regardless of node mapping. This is not the case with traditional models as the columns of their parameters are fixed, such that automorphic node permutations do not produce identical outputs. The GNN outputs a matrix of node representations, called embeddings, which contain information about each node's features, as well as the features of their k-hop neighborhood. The node embeddings are then used as inputs to a policy network that selects the best action. Importantly, actions are not formulated as a fixed-length list, as is done by prior works, including graph-based approaches (Janisch et al., 2020); instead, we represent actions as functions upon individual nodes in the graph. This allows for changing topologies and changing action spaces. Adding an additional host to a network means adding several more actions relating to the defense of that host. For tabular methods, this would require fully retraining the agent; with our method, the action space is already a function of the graph size, so no retraining is required.

This work presents a generalizable framework for graph-based RL that allows for actions upon nodes and edges, with variably sized graphs. We will show that our approach finds defense strategies that generalize better than prior work in the same field. The generality afforded by relational inductive bias means our agents can be deployed to new environments without retraining, saving potentially days of training time. In a simulated network environment, our agents score more than 4x higher than prior works and maintain that high score across environments with varied numbers of hosts– something the prior works are unable to do at all. In more complex environments, we show that when we perturb the topology or introduce new adversaries, our models perform better than all prior works. Finally, we show that our approach is also applicable to multi-agent reinforcement learning problems. Ours is currently the highest performing non-heuristic policy in the CC4 (TTCP CAGE Working Group, 2023) environment. The source code for the agents and experiments is available at `https://anonymous.4open.science/r/ACD-With-GraphRL-E543`

2 Related Work

Relational Inductive Bias: As in traditional machine learning, an overfit RL agent will optimize its policy to the random noise in the environment rather than its true distribution. This reflects a high degree of variance in the model. To counter overfitting, one must increase the model's bias (Geman et al., 1992). Inductive bias helps models constrain their search space and reason with more generalized approaches (Mitchell, 1980). Inductive bias can be created implicitly via the choice of neural architecture, or explicitly by constraining the data the model receives. Motivated by the strong arguments of Battaglia et al. (2018) in favor of relational inductive bias, as well as the numerous positive results from empirical studies (Hamrick et al., 2018; Janisch et al., 2020; Khalil et al., 2017; Wang et al., 2018) we represent the networks we wish to defend as graphs.

RL for Automated Cyber Defense: There exist many agents trained for cyber-tasks that focus on individual entities within networks, or abstract networks into individual units to defend (Eghtesad et al., 2020; Wu et al., 2018). When networks are represented as graphs, they are commonly abstracted as tabular representations for simplicity (Booker & Musman, 2020; Foley et al., 2022; Applebaum et al., 2022). Works that include graph representations often focus on finding vulner-

abilities (Piplai et al., 2022; Yousefi et al., 2018) or automated penetration testing (Gangupantulu et al., 2021; Walter et al., 2021). While many graph-based ACD agents exist in the literature, they are often constrained in unrealistic ways. Prior works, (Chai et al., 2020; Chowdhary et al., 2021; Ridley, 2018; Cam, 2020) all consider the graph structure of networks, and train DQNs to defend them. However, they all fail to consider what happens if nodes are added or removed, or even if the order in which nodes are indexed changes. More relevant to our work are the ACD approaches which highlight the importance of generalization to new topologies. One method to achieve this is aggregation. Gao et al. (2021) create a generalizable agent by creating an action space that is independent of node count and order. Instead, when an action is selected, it is applied to the node that can utilize it most efficiently according to a value function. Similarly, Gao & Wang (2021) aggregate the entirety of the network into a single entity to be defended; their observations are the network's state in aggregate across k timesteps. However, it would be difficult to directly apply either of these approaches to the more general ACD environments we evaluate, as they are both specialized for narrow threat models. The most similar prior work to ours is Doorman et al. (2022). They model the network as a graph and train a DQN to select triplets of nodes to rewire for network hardening. Their approach is specific to this one task, and as we will show, it does not generalize well to more varied action spaces, but it is notable in that it is fully inductive.

3 Background

A graph is a set of discrete objects $\mathcal{V}$ called nodes, and a set of edges $\mathcal{E} \subseteq \{(u, v) \mid u, v \in \mathcal{V}\}$ denoting relationships between them. An attributed graph is a graph with features associated with each node, $f : \mathcal{V} \to \mathbb{R}^d$. We denote these features as $\mathbf{X}$. Finally, we define node embeddings as the output of a function $\Phi(\mathcal{V}, \mathcal{E}, \mathbf{X})$ that processes the graph to encode node features and topology into a single vector for each node. This function can be transductive, which means it can only produce embeddings for nodes it has seen during training, or it may be inductive, which means it can produce embeddings for nodes it has never seen before. We will evaluate both kinds of embedding functions.

A partially observable Markov decision process (POMDP) is defined as the 7-tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{R}, \mathcal{T}, \phi, p \rangle. \tag{1}$$

Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and $p(s_0)$ is the distribution of possible starting states. $\mathcal{T}(s' \mid s, a)$ is the possibly stochastic transition function that determines the next state given a state and action. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a scalar reward function. The set $\mathcal{O}$ is the observation space, which is derived from the true state via the function $\phi : \mathcal{S} \to \mathcal{O}$. Any policy acting upon $\mathcal{M}$ will only observe the output of ϕ.

Kirk et al. (2023) define context-based POMDPs (CMDPs) as the set of all possible POMDPs in a space conditioned by a parameter sampled from distribution C. This parameter changes the possible state space, transition probabilities, and rewards, but not the action space. If we divide the context space into disjoint sets, $\mathcal{M}|_{C_{\text{train}}}$ and $\mathcal{M}|_{C_{\text{test}}}$, the objective is to find policy π which maximizes the reward over CMDPs parameterized from the test space conditioned only on experiences from the training space.

4 Graph-based Reinforcement Learning

In this section, we will describe the architecture of our graph-analytic models. We will first describe how we represent observations as graphs and actions as graph edits. Next, we describe the three models we evaluated: one transductive model, and two inductive models. Transductive models assume the size and node-ordering of a graph is fixed, while inductive models can adapt to unseen environments, with varying numbers of nodes. We used GCN as the GNN architecture, but this approach is agnostic to the specific choice of GNN.[1]

[1]Experiments showed low variance between different choices of GNN architectures. We provide an ablation study demonstrating this in Appendix A.

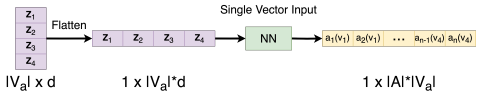

Figure 1: Transductive Actor

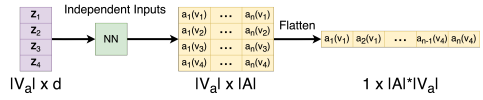

Figure 2: Inductive Actor

4.1 State-Action Space

States: In all experiments, we use individual hosts as nodes, and forms of inter-host communication as edges. Other entities within a network can also be modeled as nodes in a graph, with their node type specified as one of their node features. Unlike prior works which use features derived from graphs as their observations (Collyer et al., 2022; Wolk et al., 2022), we use the attributed graph itself. This means agents' observations are represented by the 3-tuple $\mathcal{O} = \langle \mathcal{V}, \mathcal{E}, \mathbf{X} \rangle$, representing nodes, edges, and node features, respectively. Importantly, we do not assume the agent has full visibility into the network. The graph provided in the observation may have missing information about node features (e.g., whether hosts are compromised), edges (e.g,. possible paths an attacker could use to pivot through the network), or nodes (e.g., files on the hosts), such that $\mathcal{O} \subseteq \mathcal{S}$.

Actions: Following the model of Janisch et al. (2020), we model each game as a system of objects (or nodes) that have relationships (or edges) with one another. Actions can be performed upon some subset of nodes to change the environment. Thus, for each actionable node $v_i \in \mathcal{V}_a \subseteq \mathcal{V}$, given an action space $\mathcal{A} = \{a_0, ..., a_n\}$, actions are represented as functions upon those nodes, $a_n(v_i)$. This makes the total action space for an environment $\mathcal{A} \otimes \mathcal{V}_a$. By representing actions as functions upon discrete objects, rather than a fixed array as is done by tabular methods (Watkins & Dayan, 1992), we are free to change the action space without retraining by changing the size of the set $\mathcal{V}_a$.

The results of actions produce graph edits. These edits may be in the form of node or edge additions or deletions, or changes in nodes' features. For example, the action $a_n(v_i)$ may create a node v_j with an edge to v_i, it could change the feature vector x_i associated with node v_i, or it could remove the node v_i from the graph entirely. With this abstraction, we can further extend the action space to include edge-level actions as a function of $a_n(v_i, v_j)$, upon actionable edges $\mathcal{E}_a$. This work considers environments that have node-level and edge-level actions, $\mathcal{A} = \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{E}$.

4.2 Agent Design

To combine the information about the graph structure, and its features, we use a graph neural network (GNN) (Scarselli et al., 2008) to produce node embeddings. We employ a graph convolutional network (GCN) (Kipf & Welling, 2016) for its expressiveness, and inductive abilities. The defining function of the GCN is

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\mathbf{D}^{\frac{1}{2}}\mathbf{A}\mathbf{D}^{\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}\right) \tag{2}$$

where $\mathbf{A}$ is the adjacency matrix of the graph plus $\mathbf{I}$. $\mathbf{D}$ is the degree of each node, and $\mathbf{W}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ are trainable parameters. For each node in the graph, this function averages the features of its neighbors, then passes the output through a single fully connected nonlinear neural network layer. Using k GCN layers will encode information about each node's k-hop neighborhood. In practice, we use the more efficient message-passing paradigm implemented by PyTorch Geometric (Fey & Lenssen, 2019).

The agent learns via Proximal Policy Optimization (PPO) (Schulman et al., 2017). Both the actor- and critic-networks create node embeddings $\mathbf{Z}$ using 2-layer GCNs. The embeddings are then passed through additional fully connected layers to produce the probability distribution function, or the state value estimate for the actor and critic, respectively. We evaluate three methods to convert from node embeddings to action probabilities. Using the terminology of graph representation learning, we refer to these approaches as either inductive or transductive (Hamilton et al., 2017).

Transductive models operate as prior tabular RL approaches do. As shown in Figure 1, they concatenate the node embeddings into a single, fixed-sized vector, and pass it through a traditional neural network to produce a single vector of size $|\mathcal{V}_a| \cdot |\mathcal{A}|$. This approach is not generalizable to new topologies. However, as transductive models tend to outperform inductive ones (Lachaud et al., 2022), it serves as a basis to demonstrate the maximum potential of relational bias in this domain.

For inductive models, the embeddings for each node must be processed in an order- and length-invariant way. The simplest way to do this is illustrated in Figure 2. The model passes each node embedding independently through a fully connected layer, and the output is then flattened. The actor-network uses the softmax of the flattened output to directly calculate action probabilities (e.g., $v_{i,j}$ represents the log-odds of taking action j on node i). If more actionable nodes are added to the graph, the inductive GNN will output a $|\mathcal{V} \cup \mathcal{V}'| \times |\mathcal{A}|$ dimensional matrix, which can be interpreted in the same way without retraining, thus allowing for full inductivity. The critic-network, which attempts to evaluate the value of the current global state, projects each row of $\mathbf{Z}_{\mathcal{V}_a}$ into a single dimension, then pools the batch into a single value. This method, which we refer to as the *naive inductive model* is simplistic, but we will show it is also powerful. However, it has a major drawback: node embeddings do not have the context of other nodes' states if they are greater than k hops away.

To address this, we adopt a modified form of the attention-based node pooling proposed by Janisch et al. (2020). This modification occurs in the node embedding step of the model. In each layer of the GCN, the embeddings for all nodes $\mathbf{Z}$ are calculated. Then, the embeddings of actionable nodes, $\mathbf{Z}_{\mathcal{V}_a}$, are extracted and concatenated with a global graph state vector $\mathbf{g}$. We then calculate the updated global state vector of the graph as

$$\mathbf{g}' = \mathbf{g} + \phi_g\Big(\text{POOL}_{i \in \mathcal{V}_a}\{\phi_v(\mathbf{g}, \mathbf{z}_i) \cdot \phi_a(\mathbf{g}, \mathbf{z}_i)\}\Big) \tag{3}$$

where ϕ_v, and ϕ_g are fully connected networks, and ϕ_a is a fully connected network with softmax activation. POOL represents any pooling function that is order-invariant. The original work suggests sum-pooling, but this can cause issues when testing in environments that have many more nodes than the training environment. In this work, we use mean- or max-pooling to address this problem.

The actor-network concatenates the final $\mathbf{g}$ vector to each of the final node embeddings, then uses the final vectors in the inductive method we previously described. This ensures that each node embedding contains information about all other nodes in the network, which allows them to weigh the importance of taking an action upon themselves, vs. an action somewhere else in the network. The critic network, as it is calculating the global value of the network's state, uses the $\mathbf{g}$ vector directly as input. We refer to this method as the *attention inductive model*.

Inductive models also support *edge-level actions*. We formulate the probability of taking an action on an edge as the output of an additional function that takes the source and destination node embeddings as input. In this work, to calculate the probability of taking an edge action, we calculate $f(\mathbf{Z}_{\text{src}} \odot \mathbf{Z}_{\text{dst}})$ where $f(\cdot)$ is a fully connected layer with $|\mathcal{A}_\mathcal{E}|$-dimensional output, $\odot$ represents the Hadamard product, and $\mathbf{Z}$ is the set of node embeddings such that $\langle \text{src}, \text{dst} \rangle \in \mathcal{E}_a$. The output of $f(\cdot)$ is then concatenated to the flattened probability vector of node-level actions.

5 Experiments

We evaluate our agent in three environments of increasing complexity. In each environment, our agent (the blue agent) defends the network from an attacker (the red agent). The threat model between each environment varies, but broadly, we assume that the system can be modeled as a graph of hosts whose security states transition over time between "safe" and "compromised." These transitions are governed by both attacker and defender actions. The specific transition dynamics vary by environment but are always represented within the environment's MDP. We assume that the attacker begins with root access to a single host and can take actions that increase the likelihood of compromising additional hosts. The attacker operates under partial observability of the network, must scan to discover reachable hosts, and lacks prior knowledge of the topology. The defender also has partial observability, though the scope and quality of information vary by environment. In each timestep, both attacker and defender select actions simultaneously. The attacker's objective is to maximize the number of compromised hosts over a finite time horizon, while the defender aims to minimize this quantity.

Table 1 summarizes important details of each environment. For the first experiment, we analyze the Yawning Titan environment (Collyer et al., 2022), one of the first works to frame RL-based network defense as a graph problem.[2] Next, we evaluate on the CAGE Challenge 2 (CC2) environment (CAGE, 2022), which has a more complex action space and a more fine-grained reward

[2]We select this work rather than Ridley (2018) because it has an additional focus on domain generalization.

function. Finally, we test our agent in the very complex CC4 environment (TTCP CAGE Working Group, 2023). In addition to being the largest network we test our agent on, it is also a multi-agent RL environment. Due to the page limit, please refer to Appendix B for training configuration details, and Appendix C for more details about the environments' state-action space and reward functions.

Table 1: Evaluation Environments

	Network Size	Per-node actions	Per-edge actions	Blue Agents	Red Agents
Yawning Titan	10-100	3	0	1	1
CC2	13	10	0	1	1
CC4	32-128	4	2	5	1-5

5.1 Network Simulation with Yawning Titan Environment

This environment is a two-player CMDP where a heuristic red agent attempts to spread through a computer network, and our model attempts to defend it. The computer network is represented as a random Erdős-Rényi graph (Erdős et al., 1960). Each node represents a computer, and edges represent their ability to communicate.

We use the same training configuration as the original work (Collyer et al., 2022). We train models for 5 million steps on a single random graph, then sample 50 new graphs and evaluate the models in these new environments without retraining for 10 episodes each. We compare our *Transductive*, *Naive*, and *Attention Inductive* models to the two approaches from the prior work: *Standard Observations* and *Graph Observations*. We also evaluate the architecture proposed by Doorman et al. (2022): a struc2vec-based GNN (Ribeiro et al., 2017) that uses sum aggregation to produce a global graph vector that is used in a similar way to our self-attention model. Their original approach was only designed for environments with a single action, but we modified the output to produce likelihoods for $|\mathcal{A}|$ actions per-node. Their architecture is similar to ours, but their approach uses a DQN rather than PPO. It is included here to compare the utility of both RL strategies for this problem. Table 2 reports the average reward of all 500 episodes, normalized such that 1.0 is a theoretically perfect score if no hosts were compromised at all.

Table 2: Mean score on random environments (higher is better)

	$\lvert\mathcal{V}\rvert = 10$	$\lvert\mathcal{V}\rvert = 20$	$\lvert\mathcal{V}\rvert = 40$
Standard Observations	0.2511 ± 0.017	0.2310 ± 0.013	0.1683 ± 0.010
Graph Observations	0.2090 ± 0.005	0.3037 ± 0.013	0.2047 ± 0.013
Doorman et al. (2022)	-0.1073 ± 0.009	0.2123 ± 0.008	0.1983 ± 0.006
Transductive	0.4246 ± 0.011	-0.1327 ± 0.000	-0.0966 ± 0.000
Naive Inductive	0.2681 ± 0.036	0.5439 ± 0.012	0.9308 ± 0.004
Attention Inductive	$\mathbf{0.8167} \pm 0.009$	$\mathbf{0.5634} \pm 0.012$	$\mathbf{0.9955} \pm 0.002$

Unsurprisingly, the transductive model is unable to generalize to new environments. It is likely that it overfit to the node ordering it observed during training. Both the naive and attention-based inductive models more than double the tabular methods' scores. When compared to the other GNN-based technique, as environments grew more complex, the model became less able to generalize to new graphs. Despite achieving scores comprable to our models during training, when evaluated on the new graphs, the Doorman et al. (2022) models failed to generalize. This may be explained by PPO's greater exploration ability, allowing it to find more optimal policies in more complex environments compared to DQN De La Fuente & Guerra (2024). It could also be that the prior work's choice of sum for the readout function in their model did not generalize to graphs with varied densities, leading to oversmoothing. We also observe that when $|V| = 40$, the attention-based model appears to have found a nearly perfect strategy. We hypothesize that because it was trained in a more complex environment, the observations it received during training were more varied, and forced the model to find a more general policy, while models in smaller environments became more overfit.

To further demonstrate the generalization of the policies our models learn, we evaluate the high-scoring $|\mathcal{V}| = 40$ models in environments with different numbers of nodes than they were trained in. Unlike the prior work, which cannot generalize to different environment sizes, our approach is fully inductive. As before, we evaluated the model without retraining 10 times on 50 randomly sampled

graphs for each environment size. Figure 3 plots the scores of the naive inductive and attention-based models, as well as Doorman et al. (2022) in these different environments.

Both of our models, even in the largest environment we tested, still had more than double the scores of the Euclidean models in the simplest environments. The Doorman model could generalize to smaller environments, but failed to attain comparable scores in every larger environment; this supports our claim that sum aggregation of the global vector results in unpredictable behavior in larger environments than the model was trained in. Interestingly, its performance in the smaller environments is higher than the models trained there, a trait common to all models evaluated. The attention-based model achieves higher scores overall: its strategy remains near-perfect in environments where $|\mathcal{V}| \leq 50$. However, it is more sensitive to the growing complexity of larger environments than the naive model is. While the naive model has slightly lower scores than the attention-based model, its simpler design allowed it to remain generalizable in much more complex environments. Its score also decays as complexity increases but at a much lower rate than the attention-based model. These results are likely a result of the No Free Lunch theorem (Wolpert & Macready, 1997), which states that any increase in a model's performance in one problem domain must be accounted for by a deficit elsewhere. These results seem to indicate that the cost of good model generality may be lower performance in easier regions of the environment, but consistent performance across more of the space.

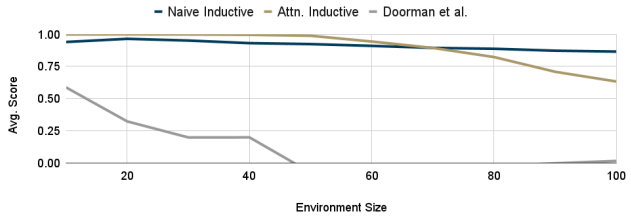

Figure 3: Scores attained by the $|\mathcal{V}| = 40$ GNN models on graphs of different sizes.

5.2 CAGE-2: A Simulated Enterprise Network

The CAGE Challenge-2 (CC2) was a reinforcement learning contest that aimed to "support the development of AI tactics, techniques, and procedures...for autonomous cyber operations" (CAGE, 2022). In this scenario, the red agent is trying to reach and compromise a specific, important host called OpServer0. CC2 provides two heuristic-based red agents: B-Line, and Meander. The B-Line agent knows the fastest route from its starting machine to OpServer0, and will take the same path of lateral movements through the network with little variation. The Meander agent is slower, performing a breadth-first search through each subnet before moving to the next subnet.

Training: All GNN models are trained using the same hyperparameters and settings. For each episode, a red agent is selected randomly with even probability. We then simulate 100 timesteps. After 100 episodes, the agent's weights are updated according to the PPO algorithm (Schulman et al., 2017). The agent only uses 4 epochs per update and otherwise uses the same hyperparameters as StableBaselines3 (Raffin et al., 2021). Both the actor and critic networks use two-layer GCNs with 256- and 64-dimensional layers.[3] The attention-pooling models use a 256-dimensional global vector with mean-pooling. All models were trained for 100k, 100-step episodes.

Default Game: Each agent was evaluated on 100 episodes with lengths $\{30, 50, 100\}$ against one of the heuristic red agents. The average score of these episodes is reported in Table 3. We compare our models to the top two performers from the original competition: *Cardiff* (Hannay, 2022), an HPPO approach which trains expert agents against the B-line and Meander red agents, and heuristically decides which model to use, and *Keeping it RL* (Wolk et al., 2022) an ensemble-of-ensembles of PPO models trained against different red agents. These models outperform ours in the competition evaluation, but not by a wide margin.

As expected, because the evaluation environment is the same as the training environment, the transductive model achieves the best average score of our models. However, the attention-based model

[3] An ablation study on this hyperparameter is provided in Appendix A.

Table 3: Scores on the CC2 Environment (smaller is better)

		30 Steps		50 Steps		100 Steps	
	Total	B-Line	Meander	B-Line	Meander	B-Line	Meander
Cardiff	**-54.57** ± 0.43	**-3.47**	**-5.64**	-6.41	**-8.69**	-13.76	-16.60
Keeping it RL	-56.90 ± 0.59	-3.48	-6.47	**-5.85**	-10.33	**-11.39**	-19.38
Transductive	-57.30 ± 0.65	-3.58	-6.25	-6.59	-9.86	-14.09	-16.94
Naive Inductive	-62.08 ± 0.63	-4.16	-7.20	-7.21	-10.90	-15.15	-17.45
Attention Inductive	-60.14 ± 0.66	-4.39	-6.99	-7.68	-9.80	-14.80	**-16.48**

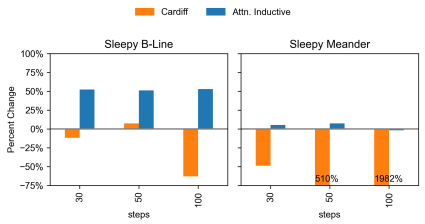

Figure 4: Evaluation against unseen red agents

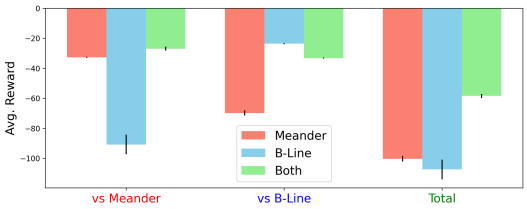

Figure 5: Evaluation of models trained against a single red agent tested against both red agents

scores almost as high in several instances, and even better in the 100-step game against Meander. Additionally, the difference in performance between the inductive and transductive models is minor.

Adversary Generalization: While the top models are very good at this specific task, they are also very brittle and overfit to the environment. In a realistic scenario, we expect the adversary to act in unpredictable ways. We create two new red agents: Sleepy-Meander, and Sleepy-B-Line. These are slower red agents; the only difference in their strategy is that at every turn they have a 50% chance of selecting no-op instead of the move the non-sleepy agent would have taken. Compare this to training against automated attacker agents, and the defense agent encountering a slower, human attacker for the first time. We evaluate our transductive model and the CardiffUni model against these new agents with no retraining. The results in Figure 4 show that this minor, realistic perturbation caused the Cardiff agent to experience a 1,982% decrease in its score in the worst case.

In comparison, utilizing relational inductive bias allows our agents to learn more generalizable policies, which is evident in how they can take advantage of the weaknesses of new attackers. For example, because the Sleepy B-Line agent is 50% slower, our model achieves a 50% better score. Against the slower meander agent, the Cardiff HPPO agent selects the expert policy for B-Line and suffers immensely. On the other hand, our agent applies its more universal policy that it learned from graph analysis and scores slightly better, or about equal to before. These results show that relational inductive bias allows for generalization to new attackers.

In another experiment, we trained our blue agents only against a single red agent before testing them against both red agents. The results of this study are shown in Figure 5. We found that against unseen agents, our models perform comparably to the HPPO agent evaluated by Wolk et al. (2022). Interestingly, the agent trained against only Meander scores slightly lower against the Meander agent, compared to the baseline agent that was trained against both red agents. These results suggest that the more general policy the default agent learned to defend against both red agents is stronger than the locally optimal policy the Meander-specialist discovered to defend against a single agent.

Domain Generalization: In addition to new adversaries, it is important to evaluate how agents behave when faced with new environments. This concern about generalizability was shared by Wolk et al. (2022), so they devised 3 new scenarios to evaluate their models in new environments. Scenarios 3 and 4 shuffle the order that hosts are indexed in two of the subnets; Scenario 5 adds paths from each machine in one subnet to two hosts in another subnet. These changes are made to the configuration file that generates the environment and are not explicitly communicated to the agents. For all generalization experiments, we evaluate agents with the parameters learned from the default CC2 environment without retraining.

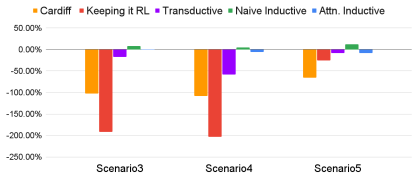

Figure 6: Change in average score under new scenarios proposed by "Keeping it RL" Wolk et al. (2022)

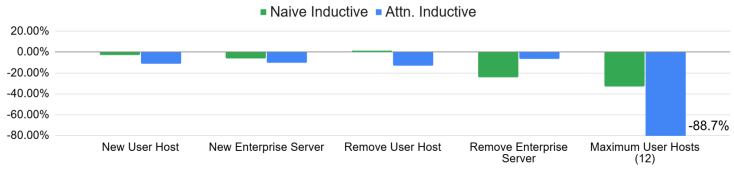

Figure 7: Change in inductive agents' scores in new environments with different numbers of hosts.

Figure 6 shows how different agents respond in these new scenarios. We note that even our transductive model out-performs the prior works, highlighting the utility of relational inductive bias. We further observe that, as the theoretical backing predicts, the inductive models are unaffected by the scenarios that change the ordering of nodes.

The scenarios proposed by Wolk et al. (2022) are a good starting point, but we feel that their changes do not go far enough. In addition to these scenarios, we also evaluate what happens when every index in the graph is perturbed. We observed that both inductive models' scores changed by $< 2\%$ for the index perturbation experiments, which was well within the standard error we observed in the previous experiments. However, the transductive model's average score dropped to -1,950.72.[4] From this, we conclude that the transductive model is also overfit.

Finally, we evaluate several new scenarios. These scenarios each involve a different number of hosts than were present in the training graph, so the tabular methods that we evaluated previously and the transductive model cannot run without retraining. Figure 7 shows how changing the environment affects the inductive agents' abilities to defend the network. In the first four scenarios, we simply add or delete a node from a subnet. These minor perturbations in the network affect our agents very little. In both instances where a host is added, the agents' scores decrease slightly, but this is expected, as the attack surface increases with every additional host. Removing an enterprise server negatively affects the naive model, while having very little effect on the self-attention model. The naive model has only a local view of each node, so it does not know when to prioritize other subnets. As a result, it rarely attempts to defend the user subnet, opting instead to defend the Enterprise subnetwork. This inability to shift its attention likely explains the score decrease.

In the last scenario, where the user subnetwork is filled, both agents have difficulty defending the network, as the attack surface is nearly doubled: 8 new users are added. Like the Yawning Titan experiments, the increase in the size of the environment affects the self-attention network more than the inductive network, which highlights the difficulty and importance of balancing good scores in simpler areas of the problem space with generality across regions with greater complexity.

5.3 CAGE-4: Multi-Agent Reinforcement Learning

The CAGE Challenge-4 (CC4) (TTCP CAGE Working Group, 2023) extends CC2 into a Multi-Agent Reinforcement Learning (MARL) task. Now, multiple agents each defend one of five segmented networks, with actions and objectives similar to CC2. One key difference for this environment is its randomly initialized topology. During each episode, subnets are generated with 1-6 servers and 3-10 user hosts. Because networks are variably sized, prior transductive and tabular-based solutions will not work.

[4]For reference, selecting actions randomly will produce a score of 1999.41.

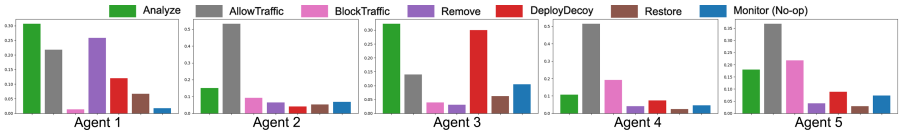

Figure 8: Action distributions of five CC4 agents. Each was trained independently, but interestingly, agents defending similar networks, e.g., Agents 2, 4, and 5 found similar strategies. Agent 1 defended the largest subnet, where it learned to analyze more frequently for better information about the large network it defended, while the latter four focused on enforcing firewall rules.

The action space for this environment is similar to CC2 with two new edge-level actions: AllowTraffic and BlockTraffic, which create or delete edges between subnet nodes. We train five independent instances of our self-attention inductive model in this environment using the same configuration as in CC2 to measure our approach's applicability to MARL tasks. Each agent independently learns a policy for the subnet it defends. We illustrate the action distribution used by each agent in Figure 8.

Our approach was the highest-performing non-heuristic agent submitted to the CC4 competition and finished in fifth place overall (Kiely et al., 2025). This result highlights the wide applicability of graph-based RL in fields beyond simple two-player games. Our agent's ability to adapt to variable network sizes and topologies allowed us to apply it to this new challenge with only minimal changes to accommodate the new action space. In a metareview of the competition, the competition runners identified the inability to adapt to the random initialization of the environment as the main detriment for other MARL approaches that were submitted (Kiely et al., 2025). Representing actions as functions upon nodes and edges allowed our approach to naturally adapt to this challenge. These results show great promise for future research into multi-agent reinforcement learning with relational inductive bias.

6 Limitations and Future Work

In this work we evaluate our approach on relatively small graphs. Scaling to enterprise-grade networks and more adaptive attackers is an important future goal. However, our primary contribution lies in the blue agent design, not the design of simulation environments or adversarial agents. Given the capabilities of existing simulation frameworks, our evaluations were limited to relatively simple networks and subsequently small graphs. As the ACD field matures and more realistic simulation frameworks emerge from ongoing efforts (DARPA, 2022), we expect our approach to transfer naturally to richer settings.

Further engineering may be required to address how the actions that agents take would be translated into real-world cybersecurity rules. Additionally, further analysis needs to be done to determine the utility of ACD agents, and how they would impact the humans using the networks they defend. In all experiments, the bottleneck for throughput is the environment, rather than the model. However, with larger networks, more optimizations may be required to process observations at a reasonable speed. Finally, we only consider actions that manifest as node-level edits, with the exception of some simple edge-level actions in CC4. Future work on efficient representation of edge- and global-level actions within a graph-based RL framework is a promising next step.

7 Conclusion

As automated cyber defense becomes a more plausible option for businesses and governments, the high cost of retraining agents is an important constraint to consider. We have demonstrated that our proposed agent can generalize and defend never-before-seen networks from attackers with novel behavior. Because the agent understands the network as a graph and views its actions as graph edits, its policy is less sensitive to environmental perturbation. We show that our graph-based approach outperforms top RL approaches in many environments by a wide margin in the face of slight and major environmental perturbations. Our results show that relational inductive bias is a powerful tool for improving agents' generalizability, and a step toward ACD in real-world systems.

REFERENCES

Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *Computer Communications*, 196:184–194, 2022.

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*, pp. 149–159, 2022.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Lashon B Booker and Scott A Musman. A model-based, decision-theoretic perspective on automated cyber response. *arXiv preprint arXiv:2002.08957*, 2020.

CAGE. TTCP CAGE challenge 2. In *AAAI-22 Workshop on Artificial Intelligence for Cyber Security (AICS)*, 2022.

Hasan Cam. Cyber resilience using autonomous agents and reinforcement learning. In *Artificial intelligence and machine learning for multi-domain operations applications II*, volume 11413, pp. 219–234. SPIE, 2020.

Xinzhong Chai, Yasen Wang, Chuanxu Yan, Yuan Zhao, Wenlong Chen, and Xiaolei Wang. Dqmotag: deep reinforcement learning-based moving target defense against ddos attacks. In *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, pp. 375–379. IEEE, 2020.

Ankur Chowdhary, Dijiang Huang, Abdulhakim Sabur, Neha Vadnere, Myong Kang, and Bruce Montrose. Sdn-based moving target defense using multi-agent reinforcement learning. In *Proceedings of the first International Conference on Autonomous Intelligent Cyber defense Agents (AICA 2021), Paris, France*, pp. 15–16, 2021.

Josh Collyer, Alex Andrew, and Duncan Hodges. Acd-g: Enhancing autonomous cyber defense agent generalization through graph embedded network representation. In *International Conference on Machine Learning*, 2022.

DARPA. Cyber Agents for Security Testing and Learning Environments (CASTLE). `https://sam.gov/opp/5fa7645fdf464f70b5c67e24585926f7/view`, Oct 2022. BAA Number: HR001123S0002.

Neil De La Fuente and Daniel A Vidal Guerra. A comparative study of deep reinforcement learning models: Dqn vs ppo vs a2c. *arXiv preprint arXiv:2407.14151*, 2024.

Christoffel Doorman, Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Dynamic network reconfiguration for entropy maximization using deep reinforcement learning. In *Learning on Graphs Conference*, pp. 49–1. PMLR, 2022.

Taha Eghtesad, Yevgeniy Vorobeychik, and Aron Laszka. Adversarial deep reinforcement learning based adaptive moving target defense. In *Decision and Game Theory for Security: 11th International Conference, GameSec 2020, College Park, MD, USA, October 28–30, 2020, Proceedings 11*, pp. 58–79. Springer, 2020.

Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

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*, pp. 1252–1254, 2022.

Rohit Gangupantulu, Tyler Cody, Abdul Rahma, Christopher Redino, Ryan Clark, and Paul Park. Crown jewels analysis using reinforcement learning with attack graphs. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–6. IEEE, 2021.

Chungang Gao and Yongjie Wang. Reinforcement learning based self-adaptive moving target defense against ddos attacks. In *Journal of Physics: Conference Series*, volume 1812, pp. 012039. IOP Publishing, 2021.

Yazhuo Gao, Guomin Zhang, and Changyou Xing. A multiphase dynamic deployment mechanism of virtualized honeypots based on intelligent attack path prediction. *Security and Communication Networks*, 2021:1–15, 2021.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

John Hannay. CardiffUni CAGE2 submission. `https://github.com/john-cardiff/-cyborg-cage-2`, 2022.

Jaromír Janisch, Tomáš Pevnỳ, and Viliam Lisỳ. Symbolic relational deep reinforcement learning based on graph neural networks. *arXiv preprint arXiv:2009.12462*, 2020.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Mitchell Kiely, Metin Ahiskali, Etienne Borde, Benjamin Bowman, David Bowman, Dirk van Bruggen, KC Cowan, Prithviraj Dasgupta, Erich Devendorf, Ben Edwards, et al. Exploring the efficacy of multi-agent reinforcement learning for autonomous cyber defence: A cage challenge 4 perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28907–28913, 2025.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Guillaume Lachaud, Patricia Conde-Cespedes, and Maria Trocan. Comparison between inductive and transductive learning in a real citation network using graph neural networks. In *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 534–540. IEEE, 2022.

Tom M Mitchell. The need for biases in learning generalizations. *Rutgers Technical Report*, CBM-TR 5-110, 1980.

Aritran Piplai, Mike Anoruo, Kayode Fasaye, Anupam Joshi, Tim Finin, and Ahmad Ridley. Knowledge guided two-player reinforcement learning for cyber attacks and defenses. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1342–1349. IEEE, 2022.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 385–394, 2017.

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

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

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

TTCP CAGE Working Group. Ttcp cage challenge 4. `https://github.com/cage-challenge/cage-challenge-4`, 2023.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

Sanyam Vyas, John Hannay, Andrew Bolton, and Professor Pete Burnap. Automated cyber defence: A review. *arXiv preprint arXiv:2303.04926*, 2023.

Erich Walter, Kimberly Ferguson-Walter, and Ahmad Ridley. Incorporating deception into cyberbattlesim for autonomous defense. *arXiv preprint arXiv:2108.13980*, 2021.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International conference on learning representations*, 2018.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Shimon Whiteson, Brian Tanner, Matthew E Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pp. 120–127. IEEE, 2011.

Melody Wolk, Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, et al. Beyond cage: Investigating generalization of learned autonomous network defense policies. *arXiv preprint arXiv:2211.15557*, 2022.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Cangshuai Wu, Jiangyong Shi, Yuexiang Yang, and Wenhua Li. Enhancing machine learning based malware detection model by reinforcement learning. In *Proceedings of the 8th International Conference on Communication and Network Security*, pp. 74–78, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Mehdi Yousefi, Nhamo Mtetwa, Yan Zhang, and Huaglory Tianfield. A reinforcement learning approach for attack graph analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 212–217. IEEE, 2018.

A Ablation Studies

In this section, we present ablation studies on the number of parameters used by the models, and the choice of GNN. For all ablation experiments, due to the length of time required to train models for 1M episodes, we constrain training time to 100k episodes. Otherwise, all parameters are fixed, and the same as they were for the original CC2 experiments other than the hyperparameter being ablated.

Hidden Dimensions. In these experiments, in addition to changing the hidden dimension, we also kept the ratio of hidden to embedding dimensions the same (4 to 1), such that the agent with a 32-dimensional hidden dimension has an 8-dimensional embedding, and the agent with 1028 hidden dimensions has a 256-dimensional embedding. Otherwise, all other hyperparameters are the same as before. We observe that as more parameters are added to the models, their average score does increase somewhat. However, adding additional dimensions comes at a cost: as models grow larger, the time it takes to perform forward and backward passes has a steep increase. Our choice in selecting 256 as the hidden dimension was to strike a good balance between score and efficiency.

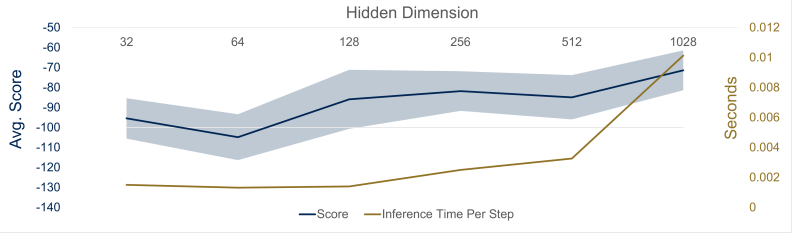

Figure 9: Average score (smaller is better) as more parameters are added to the model (standard deviation plotted as the shaded region).

Choice of GNN. To evaluate the model's sensitivity to the choice of GNN, we evaluate models trained in the CC2 environment with several different base GNNs, namely SAGE Hamilton et al. (2017), GAT Velickovic et al. (2017), GIN Xu et al. (2019). All training parameters are the same, and models are each trained for 100k episodes. We observe that each model performs roughly as well as the others. GAT has the best average performance overall, but when the standard error is taken into account, it is only better than SAGE with significance ($p = 0.04$, while $p > 0.1$ for the other two models). These results support our claim that the approach is model-agnostic; our approach is not sensitive to the choice of GNN with statistical significance.

Table 4: Scores of models with different base GNNs on CC2

		30 Steps		50 Steps		100 Steps	
	Total	B-Line	Meander	B-Line	Meander	B-Line	Meander
GCN	-80.32 $\pm$ 0.99	-5.521	-8.136	-10.718	-12.541	-21.709	-21.698
SAGE	-84.23 $\pm$ 1.05	-5.458	-9.563	-8.727	-15.069	-19.232	-26.18
GAT	-77.59 $\pm$ 1.39	-5.164	-9.404	-9.373	-14.282	-16.937	-22.434
GIN	-81.44 $\pm$ 1.05	-5.809	-8.003	-9.954	-12.802	-22.692	-22.179

B Training Configuration

To better facilitate replication of our work, in Table 5, we provide the hyperparameters used for each model. Due to the length of time required to train each model, we did not run ablation every hyperparameter, and instead opted for default settings in most cases. In the Yawning Titan environment, we selected 5M steps as the length of training time for fair comparison with the original paper. We selected the lower hidden and embedding dimension in Yawning Titan after initial tests with parameters identical to the CC2 environment performed poorly. In both environments, we tested both max and mean pooling and selected the model that performed best.

Table 5: Model Hyperparameters

	YT	CC2/4
Hidden Dimension	64	256
Embedding Dimension	32	64
Pooling Function	Max	Mean
Actor LR	3e-4	3e-4
Critic LR	1e-3	1e-3
Episode Length	600 Steps	100 Steps
Episodes per Update (N)	10	100
Training Time	5M Steps	1M Episodes

All agents were trained on a server with an Intel Xeon Gold 6338 CPU using a maximum of 100 cores to simulate N episodes in parallel. Using a GPU would accelerate training slightly, but the main performance bottleneck was the environment simulators, which were CPU-only.

C Environment Descriptions

The Yawning Titan environment represents nodes with two features each: one representing how likely an attacker is to be successful at compromising that node, and one representing whether they have been compromised. Additionally, the full list of edges in the graph representing the network is also available. We use the combination of node features and edge information as inputs to the GNNs that power our models. There are also three actions in this environment These actions are all listed and described in Table 6. In practice, however, we do not implement the Sleep action, as it is always better to Upgrade a node rather than do nothing.

Table 6: Yawning Titan Actions

Action	Description	Effect
Restore	Removes the attacker from the selected host. Simulates restoring from a previously saved image.	Update node features to "non-compromised" and reset the "vulnerability" feature to its initial value
Upgrade	Makes the selected host more difficult to compromise in the future. Simulating a software upgrade or patch.	Decrease the node's "vulnerability" feature by 0.2.
Sleep	Do nothing	

The reward function is the ratio of compromised to non-compromised hosts at each time step, with a 100-point bonus for reaching 500 timesteps without allowing every host to be compromised. If every host is compromised before step 500, the agent receives a penalty of -100 points.

The CC2 environment provides highly detailed observations with a great deal of information about each host in the network. We only considered the Host, Connection (meaning open ports), File, and Subnet entities when constructing our graphs from the available data. Table 7 contains a full description of the features considered for each node type. Entries with an asterisk denote engineered features. Entries without an asterisk are directly provided in the observations emitted by the environment.

The environment allows for ten total action types, each described in greater detail in Table 8. Unlike the Yawning Titan environment, in CC2, actions may have a cost associated with them. In this environment, only the Restore action, which fully wipes a host and restores it from an image, has a cost of -1. The Analyze action provides more information about the files on a given host. The Remove action kills any user-level shells on the host. Finally there are seven different Decoy actions, each starting a honeypot process on a different port. Certain decoys are only allowed on specific operating systems, and decoys can only be created if the host is not running any other processes on the port they would occupy.

The reward function for this environment is the number of compromised hosts in the network. For each user-type host compromised, a penalty of 0.1 points is applied; compromised server-type hosts are penalized 1.0 points; if the critical asset `OpServer0` is compromised, there is a penalty of 10 points. The full reward function is the sum of penalties from host compromises and the cost of the action selected by the agent.

Table 7: Node features in the CAGE Environments

Node Type	Feature	Feature type	Additional information
Host	Architecture	One-hot	
	OS Distribution	One-hot	
	OS Type	One-hot	
	OS Version	One-hot	
	OS Kernel Version	One-hot	
	OS Patches	One-hot	
	Is Critical*	Boolean	Applied to `Op_Server0` node in CAGE-2. Means that this node has extra penalties for being compromised
	Is User*	Boolean	
	Is Server*	Boolean	
Files	File Type	One-hot	
	File Path	One-hot	
	Version	One-hot	
	Type	One-hot	
	Vendor	One-hot	
	Density	Float	
	Signed	Boolean	
	User Permissions	int[0-7]	
	Group Permissions	int[0-7]	
	Default Permissions	int[0-7]	
Connection	Is Decoy*	Boolean	If the connection is to a decoy process running on the host
Subnet	None		Structural node to connect hosts residing in the same subnet

* Engineered features that are not provided by default

Table 8: Blue agent action space in the CAGE environments

Action	Description	Effect	Cost
Monitor	Review network traffic logs for suspicious activity. This action is taken implicitly every turn, but if selected explicitly, it functions as a no-op action.	Create edges from host nodes to connection nodes if network activity is observed.	0
Analyze	Attempt to learn it if this host has been compromised by the red agent.	Update node features if it is compromised. Add file nodes and edges connecting them to host if found during scan.	0
Remove	Attempt to remove the red agent from this machine. However, if the red agent has privilege escalated to root, this action will fail.	Update node features if successful.	0
Restore	Guarantees the red agent will be removed from this host, but causes significant disruption to the network	Remove all edges to open port nodes adjacent to the host. Reset the host node's features to show it as not compromised.	1
Decoy	Open a port on this host to act as a honeypot for the red agent. If the red agent attempts to compromise that port, it will fail. There are 7 types of decoy actions. Each may only be used if the port it would open is not in use, and sometimes only if the host uses a specific OS.	Create a new open port node with an edge to connecting it to the host.	0
AllowTraffic**	Modifies a firewall rule to allow communication between two subnets	Creates an edge between the two subnet nodes	0
BlockTraffic**	Modifies a firewall rule to disallow communication between two subnets	Deletes an edge between the two subnet nodes	0

** Actions only available in the CC4 environment.

The CC4 Environment is similar to the CC2 environment. Observations are identical between the environments with two differences. In CC4, as timesteps progress, the environment goes through three distinct phases. During each phase, there are different firewall rules and communication policies that are allowed. We communicate the current phase to the agent as input to the global vector network. The second difference is that agents are each allowed to communicate 8 bits to one another. We used this to indicate the state of each subnet the agent was defending. For each subnet, the first bit represented if any host in the subnet had been compromised. We used the second bit as a check bit to determine the difference between no message (0,0) and a message of no compromise (0,1). These messages were used as features for the subnet nodes they corresponded to. Another important distinction is that agents do not have full knowledge of the network. Each agent only receives observations about the subnets it defends; their only knowledge of subnets outside of their influence is through the messages sent by other agents.

Last, the reward structure in CC4 is different from CC2. Rewards are based on the ability of green agents (representing employees) to do either local work, or access services throughout the network. The red agent attempts to disrupt hosts, and if it is successful, when a green action tries to use it, it will fail, and the blue agent will be penalized. The exact penalty amount changes based on the current phase and the subnet of the host the green agent couldn't access.