Combining Supervised and Reinforcement Learning to Build a Generic Defensive Cyber Agent

Muhammad Omer Farooq and Thomas Kunz *

Posted Date: 19 March 2025

doi: 10.20944/preprints202503.1421.v1

Keywords: Autonomous Cyber Operations (ACOs); cyber security; blue agent; red agent; generic blue agent; machine learning

 doi:10.20944/preprints202503.1421.v1

Article

Combining Supervised and Reinforcement Learning to Build a Generic Defensive Cyber Agent

Muhammad Omer Farooq [1], Thomas Kunz [2,*]

[1] Department of Electronics and Computer Engineering, University of Limerick, Ireland
[2] Department of Systems and Computer Engineering, Carleton University. Canada
* Correspondence: tkunz@sce.carleton.ca

Abstract: Sophisticated mechanisms for attacking computer networks are emerging, making it crucial to have equally advanced mechanisms in place to defend against these malicious attacks. Autonomous cyber operations (ACO) are considered a potential solution for providing timely defense. In ACO, an agent that attacks the network is called a red agent, while an agent that defends against the red agent is called a blue agent. In real-world scenarios, different types of red agents can attack a network, requiring the blue agent to defend against a variety of red agents, each with unique attack strategies and goals. Training a blue agent that is agnostic to the type of red agent is challenging. Additionally, a generic blue agent must also be adaptable to different network topologies. This paper presents a framework for training a generic blue agent capable of defending against various red agents. The framework combines reinforcement learning (RL) and supervised learning. RL is used to train a blue agent against a specific red agent in a specific networking environment, resulting in multiple RL-trained blue agents, one for each red agent. Supervised learning is then used to train a generic blue agent using these RL-trained blue agents. Our results demonstrate that the proposed framework successfully trains a generic blue agent that can defend against different red agent types across various network topologies. The framework shows better performance compared to alternative approaches for a generic blue agent training. Additionally, to enhance the framework's generalizability, a specific type of variational auto-encoder (VAE) is integrated, further improving the performance.

Keywords: Autonomous Cyber Operations (ACOs); cyber security; blue agent; red agent; generic blue agent; machine learning

1. Introduction

The present era is unprecedented in many ways, with the rapid increase in digital connectivity transforming our society in numerous aspects. For instance, technology and digital connectivity have made remote working feasible, simplified information dissemination, enabled synchronous remote learning, and given rise to metaverse environments and online businesses. More intriguingly, the scope of digital connectivity has expanded to include connectivity among things, commonly known as the Internet of Things (IoT) [1]. IoT is becoming increasingly prevalent, playing a central role in changing how we interact with our environment. For example, autonomous and safe driving experiences are now possible because different sensors can autonomously connect to other sensor devices for information sharing and processing [2].

There is no doubt that our society is extensively digitally connected, and any disruption in this connectivity significantly impacts our daily lives. At the core of this digital connectivity is a network of interconnected computer systems. These systems not only provide connectivity but also store valuable information and grant access to it. Consider a large organization with sites in different geographical regions. Personnel interact with colleagues across multiple sites using a computer network, and the resources they need to perform their jobs may be spread across these sites, necessitating reliance on the computer network to access these resources. It is crucial that interactions occur only among intended personnel and that resources are accessible only to authorized individuals. Any malicious activity on

 doi:10.20944/preprints202503.1421.v1

such a system can lead to undesirable outcomes: information leaks, network downtime, and damage to digital resources. These scenarios can result in reputation damage, financial losses, and compromised operations, among other issues. While this example focuses on one organization, the same principles apply to any organization that relies on connected computer systems.

The above discussion highlights the importance of digital connectivity in our daily lives and underscores the necessity of securing connected computer systems to ensure continuous, effective, and efficient connectivity. Computer network security has been an active area of research for a long time, but its significance has only increased over time. This is primarily due to the emergence of new connectivity use cases [3]. Additionally, hackers are continually developing new methods to breach network security, leveraging their ability to understand the internal workings of security tools and the availability of automated scripting tools and enhanced computational power [4,5].

Recently, attackers have started using machine learning as a tool to target networked systems [6]. This presents a significant challenge to system security, as a human-based response alone may not be sufficient due to the potentially long time required to detect such attacks. Therefore, there is a need to develop automated intelligent methods that can detect and respond to attacks in a timely manner. Autonomous Cyber Operations (ACO) aim to enhance the security of computer systems through machine-based decision making. ACO offers several benefits: timely attack detection, protection of isolated systems, and safeguarding systems where trained human resources are lacking. In ACO, two types of agents are considered: blue and red. A blue agent is responsible for defending against malicious attacks, while a red agent attempts to carry out malicious activities on the system. ACO can potentially offer solutions for the complex landscape of digital threats and security challenges. By harnessing the power of artificial intelligence and machine learning, ACO enables organizations to detect, analyze, and respond to cyber threats with unmatched speed and efficiency. As cyber attacks grow in sophistication and frequency, organizations need agile defenses that can quickly adapt to new threats and vulnerabilities. Autonomous systems can analyze and learn from each encounter, continuously refining their algorithms and strategies to stay ahead of emerging threats.

Reinforcement learning (RL) has been explored in various aspects of computer network security, including penetration testing [7,8], malware detection [9,10], and anti-jamming communication systems [11,12]. Existing research demonstrates that RL can play a crucial role in networked systems' security, making it a promising machine learning paradigm for enhancing ACO. In RL for ACO, a computer network environment is modeled as a game involving blue and red agents with defined actions, network states, and rewards. This abstracted environment allows the agents to learn effective decision-making processes. However, computer networks often consist of numerous and varied entities and are dynamic, resulting in a large state space. Additionally, agents have access to a wide range of possible actions, further enlarging the state space. This makes training effective agents for ACO a challenging task. Moreover, optimizing hyper-parameters (HPs) is essential to enhance RL algorithm's performance. The large action and state spaces make HP optimization more challenging and time-consuming, as it requires numerous training runs to identify the optimal HPs.

The contribution of this paper are as follows:

- A framework for generic blue agent training capable of dealing with various types of red agents.
- An enhanced version of the framework that incorporates a custom feed-forward neural network based on the principles of variational auto-encoder (VAE) into the original generic blue agent training framework.
- An examination of the limitations of retraining blue agents against different types of red agents in an attempt to develop a generalized blue agent.
- Experimental results demonstrating the effectiveness of the proposed framework.

The rest of this paper is organized as follows: Section II presents the background and related work. Section III details the framework for generic blue agent training. Section IV provides the performance evaluation results. Finally, Section V offers the conclusions.

 doi:10.20944/preprints202503.1421.v1

2. Background and Related Work

The general categorization of machine learning methods is illustrated in Figure 1. This paper focuses on supervised learning and RL, so we will provide a discussion on these two methods.

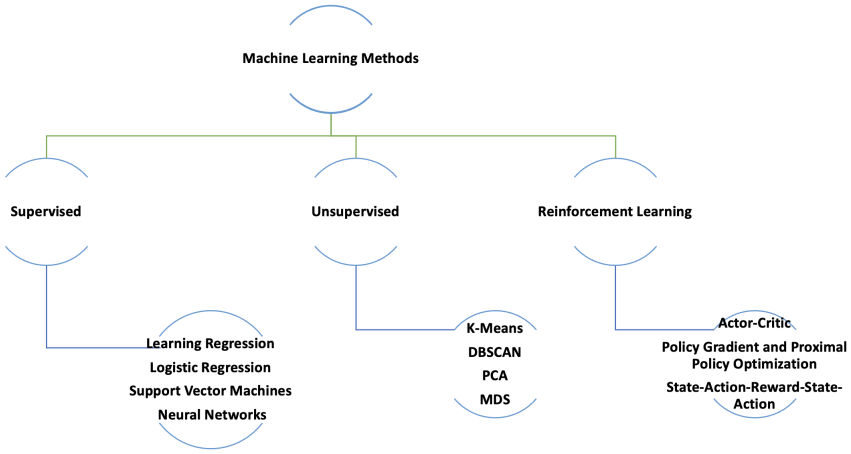

Figure 1. Categorization of Machine Learning Techniques

2.1. Supervised Learning

In supervised learning, a machine learning model is trained using an existing data set with labels. Hence, before training an effective machine learning model using supervised learning the data needs to be gathered and processed. Data must be gathered in a systematic manner as it directly effects the effectiveness and efficiency of a trained model. Data is not only required for a model training, but it is also required for testing the model. Therefore, data collection servers two purposes: model training and model testing.

There are many supervised machine learning methods, however the following are most commonly used: linear regression [13], logistic regression [14], support vector machines [15], decision trees, random forest [16], and neural networks [17].

2.2. Reinforcement Learning

RL is a machine learning paradigm [18] in which an agent/machine trains itself by interacting with an environment as shown in Figure 2. The RL setup comprises of the following: environment, agent, state, action, and reward. In RL, an agent observes the environment, capturing a state S. It then chooses one of the available actions which is applied to the environment. This leads to a change in the environment, resulting in a new state. At the same time, the environment provides an evaluation of the goodness of the action in the form of a reward. Actions that help an agent to progress towards a specific goal are rewarded positively, otherwise the reward can be zero or negative. RL does not require an existing data set to train a model, but it requires a training environment that is consistent with the target domain. For example, in order to use RL for ACO an agent needs an environment that mimics a network of computer systems, which the agent uses to train by performing actions and observing corresponding rewards. Defining RL states and actions needs domain knowledge, and the level of accuracy in defining the states and actions impacts the quality of trained agent. RL seems to be a promising choice for ACO as no prior data set is required, hence red or blue agents can learn optimal sequence of actions by interacting with the training environment. Deep reinforcement learning (DRL) represents an advancement over simple RL by incorporating neural networks into the RL framework. DRL offers several advantages over traditional RL methods [19]:

- DRL can effectively manage high-dimensional input spaces, due to its capacity to acquire hierarchical data representations using deep neural networks.
- DRL models, when trained on extensive datasets, can exhibit strong generalization capabilities to novel environments or tasks, potentially minimizing the necessity for retraining on similar tasks.

 doi:10.20944/preprints202503.1421.v1

- DRL has the capacity to acquire valuable representations of states and actions, facilitating more efficient learning and decision-making processes.

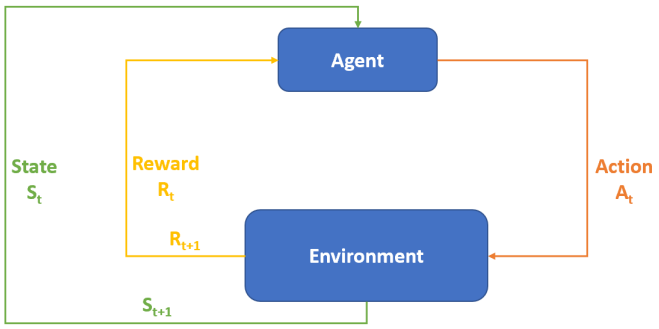

Figure 2. Reinforcement Learning Process

Typically, RL methods are classified as follows:

- Off-policy methods
- On-policy methods

2.2.1. Off-Policy Methods

Off-policy RL is a type of RL in which an agent enhances its policy using data gathered from a separate policy, referred to as the behavior policy. Essentially, the policy being optimized is called the target policy and it is different from the behavior policy. Typically, off-policy methods can use exploratory behavior policies to collect diverse data while the target policy focuses on optimal decision-making. Off-policy learning enables the reuse of past experiences, allowing the utilization of data from prior explorations or even data generated by other agents. This is the key benefit of the off-policy methods. However, off-policy methods are more complex and ensuring that they converge is a challenging task.

2.2.2. On-Policy Methods

On-policy RL entails learning and refining the same policy that is used for interacting with the environment, ensuring that the actions taken and the policy being optimized are consistently aligned. The key advantage of on-policy methods is that such methods use a unified policy, hence they are less complex compared to off-policy methods. Thus, balancing exploration (trying new actions) and exploitation (using actions that yield high rewards) is managed within the same policy framework. The major disadvantages of on-policy methods are the following:

- It may require more interactions with the environment compared to off-policy methods, as it cannot reuse data from different policies.
- Managing the balance between exploration and exploitation can present more difficulty, as the policy needs to address both aspects simultaneously.

2.3. Deep Reinforcement Learning Algorithms

Deep reinforcement learning (DRL) provides many benefits over simple RL [19]:

- DRL can effectively manage high-dimensional input spaces, due to its capacity to acquire hierarchical data representations using deep neural networks.
- DRL models, when trained on extensive datasets, can exhibit strong generalization capabilities to novel environments or tasks, potentially minimizing the necessity for retraining on similar tasks.
- DRL has the capacity to acquire valuable representations of states and actions, facilitating more efficient learning and decision-making processes.

 doi:10.20944/preprints202503.1421.v1

DRL employs neural networks to approximate either a value function or a policy function. Value-based DRL algorithms select actions that maximize the value function, with Deep Q-Networks being a prime example. Policy-based algorithms, on the other hand, focus on learning the optimal policy to maximize expected rewards. The policy gradient method exemplifies this approach. Essentially, policy-based algorithms determine a policy that maps each state to the optimal action. A key advantage of policy-based methods is their applicability to continuous action spaces.

2.3.1. Deep Q-Networks

Deep neural networks, designed to approximate the Q-function, are known as deep Q-networks (DQNs), which are value-based DRL algorithms. A key feature of DQNs at the time of their invention was their ability to directly accept raw state space as input. DQNs extend the traditional Q-learning algorithm. In Q-learning, a Q-table is maintained and updated after each action using the Bellman equation, as shown in Equation (1).

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max_A Q(S_{t+1}, A_t)) \tag{1}$$

In Equation (1), α represents the learning rate, with values ranging from 0 to 1. The first component of the equation is the old Q-value, while the second term represents the target Q-value. The discount factor for future rewards is denoted by γ. Q-learning has shown good performance when the number of states and actions is small; however, it does not scale well as the number of states and actions increases. Therefore, in scenarios with large state and action spaces, it is more effective to use a deep neural network as a function approximator to estimate the target Q-function for each possible action. In this approach, the neural network weights store the Q-table information, and each action is represented by a separate output unit. The input to the network is the state, and the output is the target Q-value for each action. The network aims to predict Q_{target} as illustrated in Equation (2).

$$Q_{taregt} = R_{t+1} + \gamma max_A Q(S_{t+1}, A_t) \tag{2}$$

Equation (3) presents the loss function used for neural network training to approximate Q_{target} for each action. In Equation (3), W represents the network parameters, which are typically optimized using the gradient descent algorithm to minimize the loss function.

$$loss = E_\pi[Q_{target}(S, A) - Q_{predicted}(S, W, A)] \tag{3}$$

2.3.2. Advantage Actor-Critic Algorithm

Advantage Actor-Critic (A2C) is a RL algorithm. It leverages both policy-based and value-based approaches by concurrently using a neural network to estimate the value function and the policy function. The A2C algorithm consists of two main components: an actor network and a critic network. Considering the current state of the environment, the actor network generates a policy. At the same time, the critic network evaluates the value of that state. The algorithm comes with the advantage function whose role is to evaluate the quality of an action in a given state. For stable and efficient training, the A2C algorithm updates the policy and value function parameters simultaneously.

2.3.3. Proximal Policy Optimization

The work presented here utilizes the Proximal Policy Optimization (PPO) RL method. Therefore, here we will briefly outline the details of the algorithm. In order to train a network, the algorithm considers small policy updates so that an agent reaches the optimal state. To control step size/policy update the algorithm uses a clip function. To initiate the PPO training process, like any other RL training algorithm, the agent is placed in an environment where it performs actions based on the input. In the early phase of training, the agent is allowed to freely explore solutions and record the outcomes. Later, after accumulating a sufficient amount of data and undergoing policy updates, the agent selects actions by randomly sampling from the probability distribution $P(A|S)$ generated by the

 doi:10.20944/preprints202503.1421.v1

policy network. The actions most likely to be beneficial have the highest probability of being selected from the random sample. After the agent transitions to a different scenario, known as a state by taking an action, it receives either a positive or negative reward. The agent's objective is to maximize its total rewards over an episode.

2.4. Reinforcement Learning Environments for Autonomous Cyber Operations

There are several RL training environments available for training agents in ACOs, such as those described in [20–23]. However, this work utilizes CybORG [24], which was released as part of the CAGE Challenge 2. The remainder of this section focuses on CybORG.

2.4.1. CybORG: An Autonomous Cyber Operations Research Gym

Inspired from OpenAI Gym [25], CybORG [24] provides tools that facilitate application of machine learning in ACO. More specifically, it provides appropriate tools and environments to train decision making models using RL in adversarial scenarios, such as, ACO. A typical learning environment in CybORG consists of a computer network that is further partitioned into different subnets. A computer network in the learning environment contains the following: different types of hosts and servers, routers, switches, and firewalls. One such scenario in CybORG is shown in Figure 3. Apart from the listed networking entities, CybORG also provides the following teams: attacker and defender: The attacker team tries to attack a network and disrupt its operation, whereas the defender team tries to defend the network against the attacker team. The attacker and defender teams are also called red and blue teams respectively. CybORG also provides different types of agents for red team. Furthermore, it also comes with a green agent whose task is to generate a network's users normal traffic.

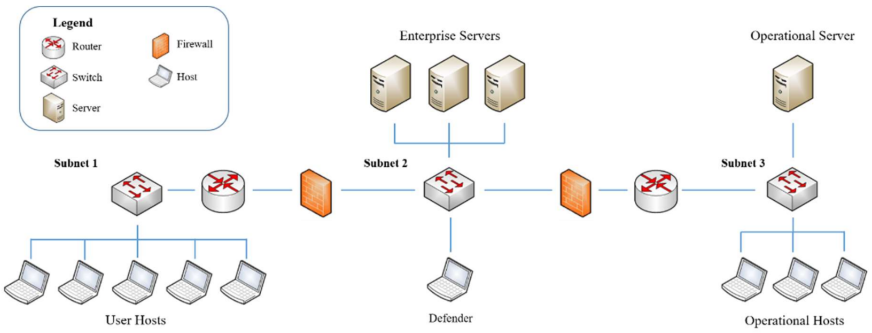

Figure 3. A Networking Scenario in CybORG [24]

CybORG comes with two distinct red agents implemented: (i) *red_meander* and (ii) *b_line*. *b_line* is also called a baseline agent. Here, functionalities of red and blue agents are explained w.r.t. Figure 3. As shown in the figure, the network consists of a user subnet, an enterprise subnet, and an operational subnet. The operational subnet stores information about logistic servers and key manufacturing. Typically, a blue agent is activated by a monitoring system, and it executes a service on each host to detect any malicious software. If the malicious software is detected, the agent removes it or if an attacker has well-established itself, the blue agent restores the system to a clean state using a backup mechanism. The red agents start the attack by accessing user hosts which are part of subnet 1. Exploitation and privilege escalation mechanisms are used by an attacker agent to access user hosts. Afterwards, servers in the enterprise subnet are exploited by the red agents. Once a server in the enterprise network is exploited, red agents obtain the IP address of the operational server to disrupt its services. The difference between the two red agents is their attack strategy. The *red_meander* agent explores the network on subnet-by-subnet basis. In each subnet it tries to compromise each host, finally reaching the operational server. On the other hand, the *b_line* agent, using prior knowledge of the network topology, tries to directly reach the operational server. The same method is used by both agents to decide on the exploitation mechanism to use on different hosts/servers. Primarily, as shown in Figure 2, RL uses state/observations, actions, and rewards concept to train an agent. Hence,

 doi:10.20944/preprints202503.1421.v1

CybORG comes with action spaces, observations, and rewards for red and blue agent. Tables 1 and 2 list some of the actions that red and blue agents can perform respectively.

A blue agent is able to maintain information about all hosts and servers in a network as part of its observation space. The blue agent's observations about each host consists of the following: (i) interface, (ii) processes, (iii) sessions, (iv) system information, and (v) user information. Interface maintains information about IP address, subnet, and interface name. The processes part of the observation space maintains information about processes' IDs and their users. Session maintains information about session owner, session ID, executing processes' IDs, and type of session. System information maintains information about a system's processor architecture, operation system and its version, and host name. Finally, user information maintains information about user groups, usernames, and passwords. Initially, the red agents observation space only consists of the above mentioned entitles for User0 in subnet 1. As part of an attack the red agents explore more about a network, hence their observation expands. Through wrappers CybORG provides capability to model observation and action spaces in a format that allows us to apply RL algorithms from existing RL libraries, such as stabebaselines and RLLIB.

In CybORG, the action space is a combination of actions and parameters. The reason is that, actions are cyber tools and they typically require parameters to execute. CybORG defines separate actions for red and blue agents. Assigning appropriate rewards to different actions in a RL environment is an important task, and it needs to be done carefully as it can have a huge impact on the RL process. For example, in the ACO scenario that is considered here, if a blue agent is not appropriately penalised for the action that cannot stop a red agent from disrupting network services, the resulting trained blue agent may not give due importance to the fact that the agent needs to stop the red agent from disrupting the network operation. Tables 3 and 4 show the rewards given to a blue agent in case of successful admin access and successful red action by a red host respectively.

Table 1. Some Actions for Red Agents

Action	**Description**
Sleep	Does nothing, and there is no parameter associated.
DiscoverRemoteSystems	It performs a ping sweep, and it requires subnet as a parameter. It returns active IP addresses on the subnet.
DiscoverNetworkServices	It is a port scan action. It requires IP address as a parameter. It returns all the open ports and their respective services.
ExploitRemoteService	It is used for a service exploit to obtain a reverse shell on a host. IP address is required as an input parameter.
EternalBlue	Used to obtain SYSTEM access over a windows-based machine.
PrivilegeEscalate	It is used to establish a privileged shell with root (Linux) or SYSTEM (Windows) privileges. For this action to succeed a user shell is required on the target host.
FTPDirectoryTraversal	Used to traverse directories that a user is not supposed to access.
SQLInjection	Action that inserts malicious database code to execute.
Impact	Represents degradation of service. It requires host name as a parameter.

 doi:10.20944/preprints202503.1421.v1

Table 2. Some Actions for Blue Agent

Action	Description
Sleep	Does nothing, and there is no parameter associated.
Analyse	Detects malware files on a host. Needs host name as a parameter.
Remove	Removes a red agent's user-level shell.
Restore	Restores a system state to a known safe state (results in network services disruption, hence large negative plenty is associated with this action).
Misinform	A decoy service action.
Monitor	Used to monitor host's state.

Table 3. Per-Turn Blue Reward for Red Admin Access

Subnet	Hosts	Blue Reward for Red Access
Subnet 1	User Hosts	-0.1
Subnet 2	Enterprise Servers	-1
Subnet 3	Operational Server	-1
Subnet 3	Operational Host	-0.1

Table 4. Successful Red Action Rewards for Blue Agent

Agent	Host	Action	Blue Reward
Red	Operational Server	Impact	-10
Blue	Any	Restore	-1

2.5. Reinforcement Learning Based Blue Agents' Training Methods

Most of the existing RL-based blue agents for ACOs can be categorised as follows [26]:

- Single agent
- Hierarchical agent
- Ensembles

Single agents are straightforward, involving the training of one agent using RL to defend against various types of red agents, and deploying only one blue agent in a networked system [27]. Hierarchical agents are more complex, typically including an agent that selects another agent to provide an action based on the type of red agent attacking the network [28]. Similarly, ensemble approaches involve training multiple agents, each receiving the network's state and providing an action based on that state. The most commonly reported action is ultimately executed. Several existing blue agents based on ensemble approaches are discussed in [26]. For existing work in ACO, the following conclusions can be drawn as per [26].

- Hierarchical agents generally perform the best, followed by ensemble agents, and then single agents.
- The PPO algorithm has been noted for its superior performance.
- Overall, RL-based methods show better generalization capabilities compared to non-RL-based approaches.

2.6. Discussion

Training generalized blue agents to support ACO is a challenging task, but its importance has driven several efforts to develop RL methods that can produce such agents. Hierarchical RL methods have shown superior performance compared to ensemble-based RL and single-agent RL methods. However, hierarchical methods require training and deploying multiple blue agents within a networked system, which is also true for ensemble-based RL methods. This makes these approaches relatively resource-intensive and can result in longer real-time decision-making processes due to the

 doi:10.20944/preprints202503.1421.v1

involvement of multiple agents. Therefore, there is a need to further explore the possibilities of training a blue agent that not only possesses generalized capabilities but also requires only a single agent to be deployed in the networked system.

3. Framework for Generic Blue Agent Training

To support ACO, we present the details of the proposed framework. This framework integrates reinforcement learning (RL) and supervised machine learning to train a single generalized blue agent capable of defending against various red agents. Figure 4 illustrates the training framework for the generalized blue agent. The components of this framework are discussed below. The framework consists of the following components:

- HP tunning for training a blue agent against a red agent using RL.
- Training blue agents against red agents using RL.
- Collecting data with the trained blue agents.
- Randomizing the collected data.
- HP tuning for supervised learning to train a generalized blue agent.
- Training a generalized blue agent using a supervised machine learning algorithm.
- Testing and deploying the generalized blue agent.

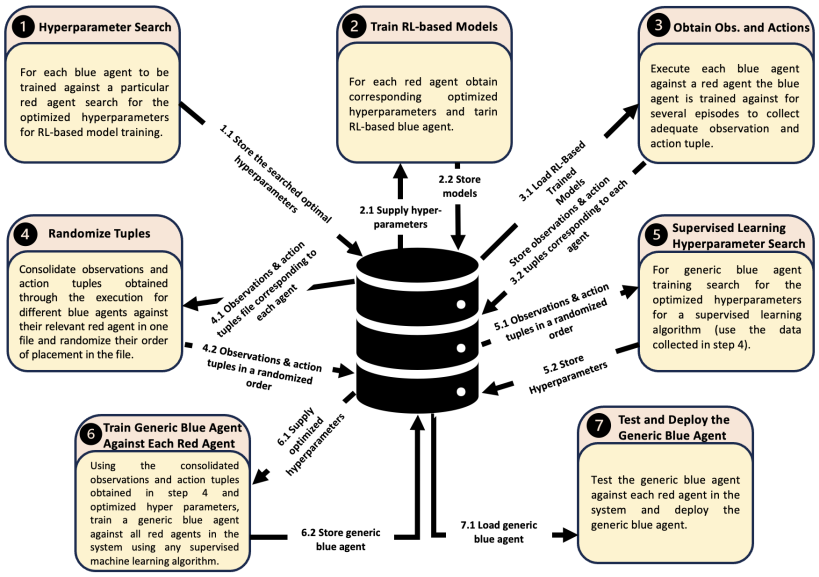

Figure 4. Proposed Framework for Generalized Blue Agent

3.1. Framework's Component Details

3.1.1. Hyper-Parameter Tuning for Blue Agent Training Using Reinforcement Learning

In the proposed framework, initially, a separate blue agent must be trained against each possible red agent. If there are N red agents in a system, N blue agents need to be trained, with each blue agent tailored to a specific red agent. In machine learning, it is well-known that tuning HPs can enhance a model's performance. For HP tuning, a range of values is explored for the different HPs associated with a given RL algorithm. The following strategy is used for HP tuning:

- Create an RL-based learning environment that includes various entities (hosts, servers, firewalls, switches, routers, etc.) along with a blue agent and the corresponding red agent.
- Provide a range of values for all possible HPs.
- Utilize an HP tuning library for RL, such as Optuna [29], to find the best HPs for training the blue agent against a red agent.
- Store the optimal HP values obtained in the previous step.
- Repeat the above steps for each blue agent that needs to be trained.

 doi:10.20944/preprints202503.1421.v1

3.1.2. Reinforcement-Learning-Based Blue Agent Training

After identifying the optimal HPs for each blue agent, the agents must be trained against their respective red agents. This training requires creating a learning environment that simulates a real network scenario, including both blue and red agents. The RL algorithm used for training a blue agent should employ the optimal HPs identified for that specific red agent. The blue agent is then trained over a large number of RL training steps against the red agent. This process needs to be repeated for the training of each blue agent.

3.1.3. Data Collection - Observations and Action Tuples

The proposed framework employs supervised learning to train a generic blue agent for ACO. Since supervised learning requires training data, the framework includes a data collection mechanism. Initially, a separate blue agent is trained against each red agent using RL, allowing the blue agent to learn the best possible actions based on the current network state/observations against its respective red agent. Similarly, the generic blue agent needs to learn optimal actions based on various network states and also identify the type of red agent attacking the network.

To train the generic blue agent, the framework collects a sufficient number of observation-action tuples by running scenarios where each trained blue agent defends against its corresponding red agent. This process should be repeated for all trained blue agents as described in Section 3.1.2. The goal is to gather enough observation-action tuples from each trained blue agent to compile a comprehensive dataset. This dataset can then be used to train a generic blue agent capable of identifying the type of red agent attacking the network and learning to take appropriate actions. The collected observation-action tuples should be stored for further processing. Algorithm 1 outlines the data collection process.

Algorithm 1: Observations and Action Tuples Collection

```
Input: blue_agents, red_agents;
Output: list_of_obs_actions_files ;
list_length = len(blue_agents) ;
i = 0;
for i < list_length do
    new_tuples_file = create_file() ;
    i++;
    episode = 0;
    env = create_aco_env(blue_agents[i], red_agents[i]) ;
    model = load_trained_model(blue_agents[i], red_agents[i]) ;
    terminated = False;
    obs = model.reset(env) ;
    for episode < MAX_EPISODES do
        while not terminated do
            action = model.predict(obs) ;
            prev_obs = obs;
            obs, reward, terminated, info = env.step(action) ;
            new_tuples_file(prev_obs, action) ;
        end
    end
    list_of_obs_actions_files.add(new_tuples_file) ;
    new_tuples_file.close() ;
end
return list_of_obs_actions_files;
```

 doi:10.20944/preprints202503.1421.v1

3.1.4. Randomizing Sequencing of Collected Data

The data collection module gathers data sequentially, with each blue agent being tested against the red agent it was trained to counter, in order to collect observation and action tuples. To prepare the data for supervised learning, all collected data must be consolidated into a single source file. Once stored, the data sequence needs to be randomized; otherwise, records for each blue agent will appear in order, which could negatively affect the performance of the supervised machine learning algorithm. For instance, a neural network might not train effectively for blue agents whose data is at the beginning of the file. Figure 5 illustrates the process of data preparation to train a generic blue agent using supervised learning.

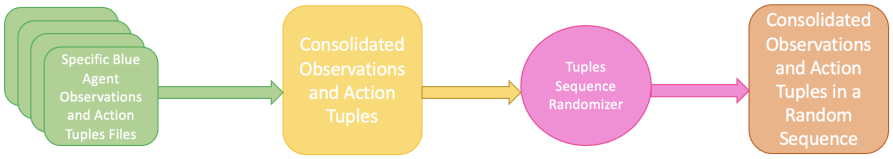

Figure 5. Preparing Data for Supervised Learning

3.1.5. Supervised Learning - Hyper-parameter Optimization

Once the data is prepared for training a generic blue agent using supervised learning, the next step is to use the data to find the optimal HPs for a suitable supervised machine learning algorithm, such as multi-layer perceptrons, support vector machines, or random forests. Depending on the chosen algorithm, different HPs need to be tuned, such as the number of hidden layers, activation function, and learning rate for a multi-layer perceptrons. Existing machine learning libraries like SciKit-Learn provide APIs for HP tuning, such as GridSearchCV and BayesSearchCV. The process for finding the optimal HPs can be summarized as follows:

- Select a suitable supervised machine learning algorithm.
- Choose a reasonable range of values for the different HPs associated with the selected algorithm.
- Provide the supervised machine learning method, the list of HPs and their value ranges, and the number of cross-validation folds to an API that searches for the best HP values.
- Use the selected library's API, such as the fit method in SciKit-Learn, to obtain the best HP values.

3.1.6. Training a Generic Blue Agent

By now, most of the necessary components for training a generic blue agent using a supervised machine learning algorithm are in place, thanks to the preceding modules of the framework. These components include: (i) training data, (ii) data pre-processing, and (iii) optimized HP values for the selected supervised machine learning algorithms. However, testing data is also needed for evaluating the trained blue agent. To address this, the framework proposes partitioning the collected data into a training set and a testing set, typically allocating 80% for training and 20% for testing. Given that the data is collected using blue agents trained through RL, a substantial amount of training and testing data can be gathered. The generic blue agent is then trained using the training data set and the optimized HP values with a supervised machine learning algorithm.

3.1.7. Test and Deploy Generic Blue Agent

After training the generic blue agent, it must be tested using the test data set. If the agent's performance is acceptable after testing, it can be deployed on a real system. In the real system, the agent periodically receives the system's state and suggests an appropriate action based on this state. If necessary, the action is executed by a script.

 doi:10.20944/preprints202503.1421.v1

3.2. Custom Variational Auto Encoder for Generic Blue Agent Training

The supervised learning component of the proposed framework uses data obtained from interactions between blue agents and red agents. This data is collected during scenarios where blue agents counteract red agents in specific settings, such as a network topology. Even slight changes in network topology can alter the observation vector. Additionally, since the framework's supervised learning part uses data collected from different types of red agents, different observation vectors might be mapped to the same action. This can negatively impact the framework's performance. To address this, we extend the proposed framework by incorporating a method to map observation vectors to a particular distribution, aiming to handle variations in observation vectors and minimize instances where the same observation vector is mapped to different actions. This method is a variation of a variational autoencoder (VAE) and is referred to as VAE variant, or VAE-V for short.

In VAE, an input data is typically mapped to a normal distribution with the aim to limit the values that latent variables can take. This process is stochastic in nature as the values taken by latent variables are sampled from a normal distribution. In the context of a generic blue agent training, once the blue agent is trained it is meant to carry out actions. In our autonomous cyber system, actions are represented as discrete numbers. In a nutshell, in the autonomous cyber system the generic blue agent expects a vector of observations and outputs a discrete number corresponding to a certain action that needs to be executed. Hence, the problem is exactly analogous to a classification problem. Here, a custom VAE is designed that handles the classification problem in the context of ACO. The following steps summaries the VAE-V functionality:

- Each observation tuple is mapped to two latent vectors of dimension N representing mean (μ) and log variance (σ) of a standard normal distribution.
- For each latent variable a point is sampled from a standard normal distribution using corresponding μ and σ.
- Latent variables are feed to the decoder to produce an action for a blue agent (the output of this VAE-V is a scalar value).

Figure 6 shows the designed VAE-V model for generic blue agent training. The encoder part of the model has an input layer $(input\_2)$, two dense layers $(dense\_5)$ and $(dense\_6)$, and two further dense layers (z_mean) and (z_log_var) to produce a latent vector of dimension 18 (latent dimension of 18 is chosen after hyper-parameter optimization).The input layer expects a vector of length 52 (in CybORG, the observation tuple consists of 52 items). The decoder part of VAE-V comprises of two dense layers $(dense\_7)$ and $(dense\_8)$, and an output layer $(dense\_9)$. It should be noted here that the number of units in $dense\_7$ and $dense\_8$ are exactly the same as in $dense\_6$ and $dense\_5$ respectively. The number of units in the output layer $(dense\_9)$ are 145 (the total number of actions defined for blue agents in CybORG). As VAE-V is designed for a classification problem in the context of ACO, the $softmax$ activation function is used in the output layer to select an action with the highest probability. The VAE-V is trained by tuning the latent space dimension.

 doi:10.20944/preprints202503.1421.v1

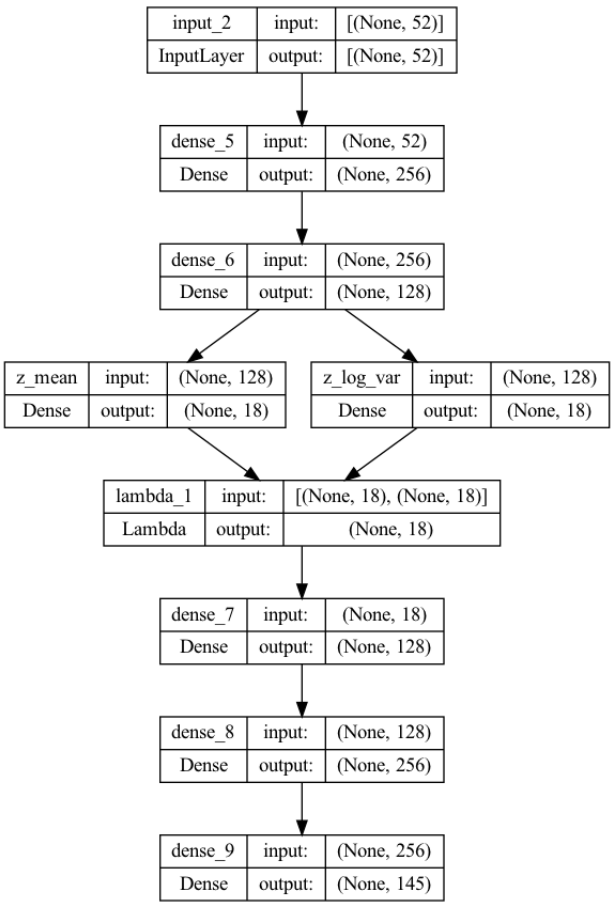

Figure 6. Custom VAE Model for Generic Blue Agent Training

4. Performance Evaluation

For performance evaluation, the following types of agents are used:

- A blue agent specifically trained against the *b_line* red agent, referred to as *BL*.
- A blue agent specifically trained against the *red_meander* red agent, referred to as *RED*.
- *BL* retrained against the *red_meander* red agent, referred to as *BL-RED*.
- *RED* retrained against the *b_line* red agent, referred to as *RED-BL*.
- A proposed framework-based blue agent that uses a Multilayer Perceptron approach, referred to as *MLP*.
- A proposed framework-based blue agent that uses a Support Vector Machine approach, referred to as *SVM*.
- A proposed framework-based blue agent that uses VAE-V, referred to as *VAE-V*.
- A hierarchical super agent, a blue agent that first determines the kind of attacking red agent, and then launches the specifically trained blue agent for that particular red agent. Variants of the hierarchical blue agent that misclassify the attacking agent by 5%, 10%, 15%, and 20% are referred to as *HA*5, *HA*10, *HA*15, and *HA*20, respectively.

The proposed framework integrates RL with supervised learning. To gather data for supervised learning, each trained blue agent is run against the red agent it was trained to oppose, collecting observation and action tuples for further learning. For each agent, 10,000 tuples were collected. We employ multi-layer perceptron and support vector machine algorithms for supervised learning, though the framework is adaptable to any supervised learning algorithm. HP optimization for both multi-layer perceptron and support vector machine was conducted using Bayesian search. For *SVM*, the HPs considered were *C*, *gamma*, and *kernel*, with optimal values found to be 10.9975, $4.501669e-06$, and *linear* respectively. For *MLP*, the HPs considered were the number of hidden layers, activation function,

 doi:10.20944/preprints202503.1421.v1

solver, learning rate, and alpha, with optimal values determined to be 10, *tanh*, *lbfgs*, *invscaling*, and 0.01542674 respectively. For VAE-V, the *ReLU* activation function is used at each layer, except for the output layer, which uses the *Softmax* activation function. The loss function is a combination of categorical cross-entropy and Kullback-Leibler divergence. The latent vector comprises of 18 elements.

We deployed our approach in CybORG, which features two distinct red agents/attackers. Each blue agent used for performance evaluation is trained for 100,000 steps. For the various performance evaluation scenarios considered here, Proximal Policy Optimization (PPO) is utilized as the RL algorithm, as previous studies have shown its superior performance and generalization compared to other RL algorithms like DQN [5]. The agents are trained using the networking scenario illustrated in Figure 3. In each scenario, every trained blue agent is evaluated over 100 episodes against the red agents.

Table 5. Hyper-Parameters and Values' Range

Hyper-Parameter	**Range of Values**
Batch Size	[8, 16, 32, 64, 128, 256, 512]
No. of Steps	[8, 16, 32, 64, 128, 256, 512, 1024, 2048]
Gamma	[0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999]
Learning Rate	log range(1e-5, 1)
Entropy Coefficient	log range(0.00000001, 0.1)
Clip Range	[0.1, 0.2, 0.3, 0.4]
No. of Epochs	[1, 5, 10, 20]
GAE Lambda	[0.8, 0.9, 0.92, 0.95, 0.98, 0.99, 1.0]
Max Gradient Norm	0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5]
VF Coefficient	[0, 1]
Activation Function	[tanh, relu, elu, leaky relu]
Neural Network Architecture	[(pi=[64, 64], vf=[64, 64]), (pi=[256, 256], vf=[256, 256])]

Table 6. Tuned Hyper-Parameter Values for Blue Agents

Hyper-Parameter	**b_line Red Agent**	**Red Meander Agent**
Batch Size	8	256
No. of Steps	128	1024
Gamma	0.9	0.95
Learning Rate	0.00018937	0.00373109
Entropy Coefficient	1.032461073e-05	0.017615274
Clip Range	0.3	0.3
No. of Epochs	10	10
GAE Lambda	0.8	0.8
Max Gradient Norm	0.5	5
VF Coefficient	0.09187	0.73582
Activation Function	relu	tanh
Neural Network Architecture	pi=[256, 256], vf=[256, 256]	pi=[64, 64], vf=[64, 64]

Our performance evaluation experiments utilize two different episode lengths: 100 steps and 50 steps. By varying the episode length, we aim to assess the trained agents' performance over both shorter and longer periods. This approach helps to highlight how the trained blue agents perform initially and how they adjust their behavior as the scenario progresses. It should also be noted that blue agents do not receive rewards but are penalized at each step. They incur heavier penalties for successful red agent actions. Therefore, an effective blue agent is one that successfully prevents the red agent's actions, resulting in a total reward closer to zero.

 doi:10.20944/preprints202503.1421.v1

4.1. Results Based on Training Topology

Figure 7 displays the mean total reward along with 95% confidence interval (CI) for different blue agents against *b_line* and *red_meander* in 100-step episodes. When the network is attacked by *b_line*, agents trained using the proposed framework show better mean rewards compared to other agents, except BL. Notably, our VAE-V agent achieves the highest mean reward, even better than BL, the agent specifically trained for *b_line*. However, this difference is not statistically significant. The RED agent trained specifically for *red_meander*, performs poorly against *b_line*, and this difference is statistically significant. Similarly, BL-RED and RED-BL perform worse against *b_line*. BL-RED performs worse than RED-BL because RED-BL is a retrained agent for *b_line* that was initially trained for *red_meander*. Consequently, it demonstrates a better mean reward compared to BL-RED, which was originally trained for *b_line* and then retrained for *red_meander*. This indicates that the agent forgets features of the *b_line* red agent while retraining. All HA agents show lower mean rewards compared to BL, but the differences in performance are not statistically significant.

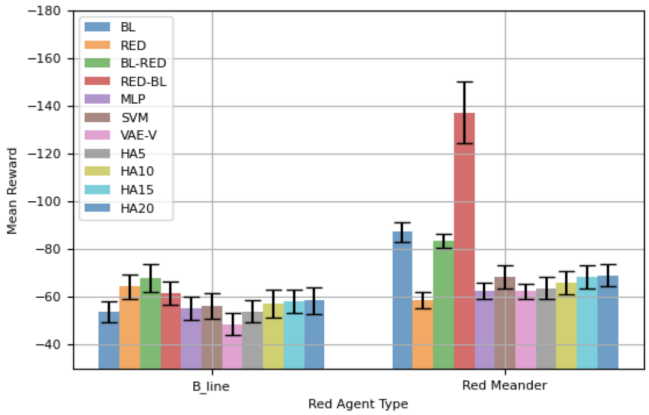

Figure 7. Performance Comparison Using Training Network Topology - 100-Step Episode

For agents' performance against *red_meander*, the RED agent achieves the best mean reward as it is specifically trained for *red_meander*. VAE-V and MLP agents, trained using the proposed framework, show similar performance to RED. BL, BL-RED, and RED-BL perform worse compared to RED, with the difference being statistically significant. Notably, RED-BL exhibits a lower mean reward compared to BL-RED, highlighting that retraining an agent leads to forgetting features of the red agent it was originally trained against. HA agents also perform worse compared to RED, and as the error rate in selecting the correct blue agent increases, the performance difference becomes statistically significant between HAs and RED. VAE-V shows a better mean reward compared to HAs, and MLP also demonstrates a better mean reward compared to some HAs.

Table 7 shows the total mean reward demonstrated by different agents considering both red agents. MLP and VAE-V exhibit superior total mean rewards, with SVM also outperforming most other agents. The proposed framework trains a single blue agent to handle both *b_line* and *red_meander*. The results in Table 7 indicate that agents trained using the proposed framework possess generalizability, enabling them to effectively handle both types of red agents.

 doi:10.20944/preprints202503.1421.v1

Table 7. Total Mean Reward - 100-Step Episode

Agent Type	BL	RED	BL-RED	RED-BL	MLP	SVM
Total Mean Reward	-140.88	-122.96	-151.33	-198.75	-117.70	-124.61
Agent Type	VAE-V	HA5	HA10	HA15	HA20	
Total Mean Reward	**-110.90**	-117.60	-123.10	-126.60	-127.43	

Previously, all results were based on episodes of 100 steps. Here, we present results from another set of experiments with an episode length of 50 steps. Prior findings have shown that retraining agents is counterproductive, so retrained agents are not considered in this analysis. Figure 8 shows the mean total rewards along with 95% CIs for different blue agents. The performance of the blue agents against *b_line* is similar to that shown in Figure 7(a). VAE-V demonstrates the best mean reward, followed by MLP and SVM. However, the performance difference between the blue agents trained using the proposed framework and BL is not statistically significant. The RED agent shows the worst performance against *b_line* as it is specifically trained against *red_meander*. Most HAs also demonstrate worse mean rewards compared to BL and the blue agents trained using the proposed framework, with the differences in performance for VAE-V and HA20 being statistically significant. The blue agents' performance against *red_meander* is again similar to what is shown in Figure 7(a). The RED agent demonstrates the best performance, with VAE-V and MLP showing comparable performance to RED, as the differences are not statistically significant. BL performs the worst since it's trained for *b_line*. The mean reward for HAs decreases as the percentage error in selecting the appropriate blue agent increases, although their performance is generally similar to RED.

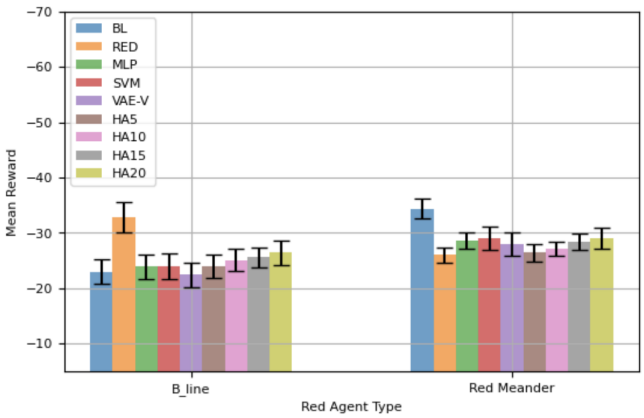

Figure 8. Performance Comparison Using Training Network Topology - 50-Step Episode

Table 8 lists the mean total rewards as demonstrated by the blue agents against both red agents. Among the blue agents trained using the proposed framework, VAE-V achieves the highest total mean reward. HA5 achieves the same total mean reward, while MLP and SVM also exhibit comparable total mean rewards. The results in Table 8 indicate that agents trained with the proposed framework exhibit a certain degree of generalizability, enabling them to effectively handle both types of red agents using single blue agent.

 doi:10.20944/preprints202503.1421.v1

Table 8. Total Mean Reward - 50-Step Episode

Agent Type	BL	RED	MLP	SVM	VAE-V
Total Mean Reward	-57.4	-58.83	-52.39	-52.92	**-50.4**
Agent Type	HA5	HA10	HA15	HA20	
Total Mean Reward	**-50.4**	-52.3	-54.0	-55.4	

4.2. Varying the Network Topology

To delve deeper into the generalization capability of the proposed framework, we examine it using network topologies distinct from the network topology employed to train blue agents.

4.2.1. Topology 2 Results

Figure 9 depicts the network topology utilized for assessing the efficacy of various blue agent training methods. Henceforth, this topology will be referred to as "Topology 2". Compared to the original topology used to train blue agents, the difference here is that *UserHost3* and *UserHost4* have been moved from *subnet1* to *subnet2*. This change should not significantly affect the existing red agents, *b_line* and *red_meander*, for the following reasons: (i) *b_line* knows the previous network topology to reach critical resources and impact the operational server, and moving UserHost3 and UserHost4 does not affect the route to the operational server. (ii), *red_meander* systematically discovers the network topology, so its behavior should not be significantly impacted. Consequently, no changes were made to the existing red agent scripts. The trained blue agents using the proposed framework should be the ones potentially affected by this change, as the number of hosts in one subnet has decreased while it has increased in the other.

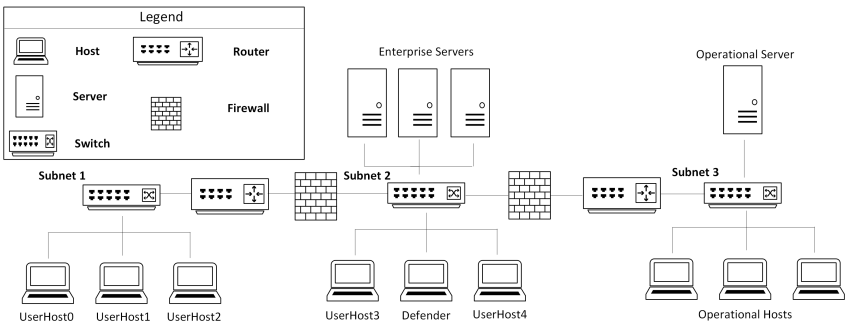

Figure 9. Network Topology 2

Figure 10 shows the mean reward achieved by different blue agents against the red agents. The results in Figure 10 are collected over 100 episodes, each consisting of 100 steps. Against *b_line*, VAE-V has demonstrated the highest mean reward among the evaluated blue agents, followed by SVM, MLP, BL, and HA5. However, the differences in performance are not statistically significant. The RED agent has shown the worst performance. Against *red_meander*, VAE-V again achieved the best mean reward, followed by SVM, RED, and HA5. Similarly, the performance differences are not statistically significant. BL performed the worst against *red_meander*.

 doi:10.20944/preprints202503.1421.v1

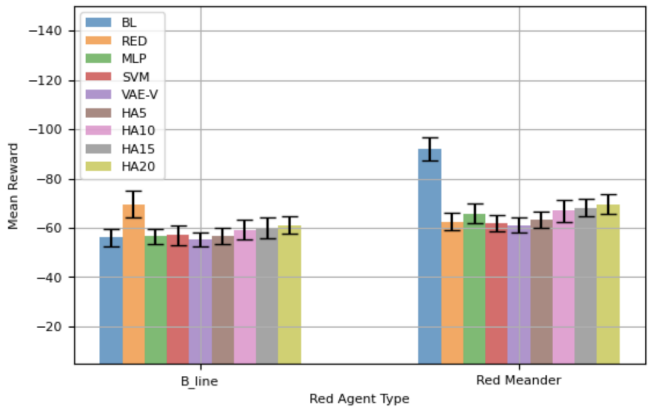

Figure 10. Performance Comparison Using Network Topology 2 - 100-Step Episode

Table 9 lists the total mean reward demonstrated by the blue agents considering both red agents, showing that VAE-V achieves the highest total mean reward, followed by SVM, HA5, and MLP. Comparing Tables 7 and 9 reveals the following:

- The performance of all blue agents trained solely with Reinforcement learning (BL, RED, and all HAs) has deteriorated in this modified environment.
- The performance of the single blue agent trained with the proposed framework either deteriorated (MLP and VAE-V) or, in the case of SVM, actually demonstrates slightly improved total mean reward.

Table 9. Total Mean Reward - 100-Step Episode (Topology 2)

Agent Type	BL	RED	MLP	SVM	VAE-V
Total Mean Reward	-148	-132.0	-122.2	-119	**-116.45**
Agent Type	HA5	HA10	HA15	HA20	
Total Mean Reward	-120.08	-126.14	-128.32	-130.5	

In summary, RL-XGBoost outperformed other defensive agents against the $red_meander$ agent, while most defensive agents trained using the proposed framework exhibited comparable performance against the b_line agent when evaluated on a different network topology from the one used during training. Overall, RL-XGBoost demonstrated the best performance. However, the performance of BL, RED, and hierarchical agents showed a more pronounced decline. These findings indicate that the proposed framework for defensive agent training possesses a degree of generalizability to variations in network topology, with RL-XGBoost particularly excelling in this regard.

Figure 11 presents the mean reward demonstrated by the evaluated blue agents for a shorted episode length, i.e., 50-step. Overall, results are similar to 100-step episode, i.e., VAE-V demonstrates the best performance followed by HA5, MLP, and SVM. BL and RED demonstrate the worst performance as also shown in Table 10. It is important to note here that VAE-V demonstrates statistically better mean rewards compared to RED against *red_meander*. Comparing agents' performance for 50-step episodes using Topology 2 with topology used for agents' training reveals that the performance of blue agents trained through the proposed framework has improved against the *red_meander* agent, but has slightly deteriorated against the *b_line* agent. Most of the HAs performance has slightly deteriorated in this case.

 doi:10.20944/preprints202503.1421.v1

Table 10. Total Mean Reward - 50-Step Episode (Topology 2)

Agent Type	BL	RED	MLP	SVM	VAE-V
Total Mean Reward	-61	-61.1	-54.43	-55.4	**-50.3**
Agent Type	HA5	HA10	HA15	HA20	
Total Mean Reward	-50.38	-52.69	-54.9	-56.40	

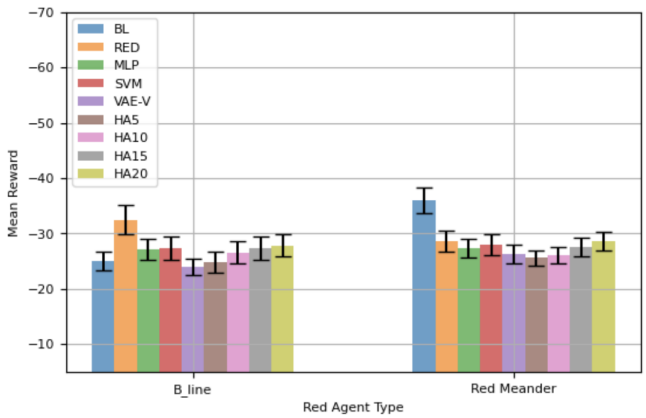

Figure 11. Performance Comparison Using Network Topology 2 - 50-Step Episode

4.2.2. Topology 3 Results

Figure 12 shows another network topology utilized for assessing the efficacy of various blue agent training methods. Hereafter, this topology is referred as "Topology 3". In Topology 3, *UserHost0* is moved to the *enterprise* subnet. Since red agents access the network through *UserHost0*, this change can impact the performance of the *b_line* red agent, as its information about the network topology is now inaccurate. However, *red_meander* discovers the network topology as part of the attack, hence with this change its overall approach should not be impacted. In general, the red agents are rule-based and they consider the host while they decide about the next action to take, hence there performance should not be significantly impacted with this change. Additionally, the host that both red agents use to access the network is now closer to their target, so the red agents may not perform poorly or could even perform better. Topology 3 is more challenging for the trained blue agents because the red agents now need fewer actions to compromise the operational server. Therefore, Topology 3 can provide more insights into the generalizability of the trained blue agents with regard to network topology variations.

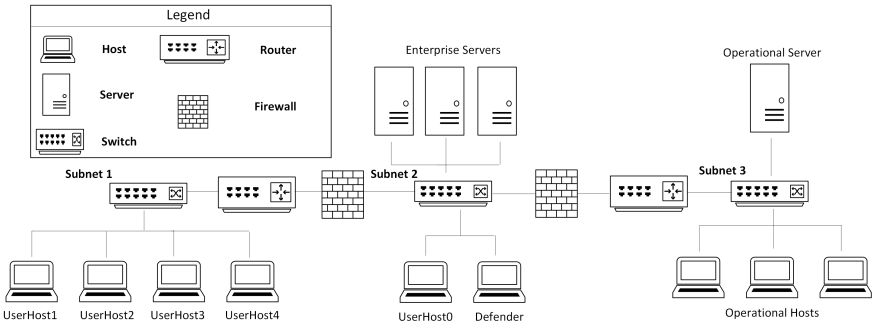

Figure 12. Network Topology 3

Figure 13 presents the performance of various blue agents against *b_line* and *red_meander* over Topology 3. Among all the trained blue agents, MLP showed a statistically significantly higher mean reward compared to all other blue agents evaluated in this study. Notably, all blue agents trained

 doi:10.20944/preprints202503.1421.v1

using the proposed framework demonstrated a statistically significantly better mean reward compared to BL, the blue agent specifically trained for *b_line*. All variants of HAs showed a higher mean reward compared to BL and RED; however, this difference was not statistically significant. Against *red_meander*, SVM and VAE-V exhibited statistically significantly better performance compared to RED, the blue agent specifically trained for *red_meander*. Furthermore, SVM showed statistically significantly better performance compared to all other evaluated blue agents in this scenario, while MLP demonstrated a comparatively poor performance, albeit slightly better than the blue agents trained solely via RL. Overall, against both red agents, the blue agents trained through the proposed framework showed a better mean reward compared to all other evaluated blue agents. Two out of three blue agents trained with the proposed framework achieved a statistically significantly higher mean reward compared to pure RL blue agents. Table 11 shows the total mean reward exhibited by the blue agents while considering both red agents. SVM exhibits the highest mean reward followed by VAE-E and MLP.

Table 11. Total Mean Reward - 100-Step Episode (Topology 3)

Agent Type	BL	RED	MLP	SVM	VAE-V
Total Mean Reward	-133.3	-136.61	-131.10	**-117.55**	-123.24
Agent Type	HA5	HA10	HA15	HA20	
Total Mean Reward	-131.74	-136.64	-138.01	-139.93	

Comparing Figure 7 with Figure 13 reveals the following about the trained blue agents' performance against the red agents:

- When the attacking red agent is *b_line*, BL and VAE-V demonstrate similar performance in both topologies. SVM, VAE-V, RED, and all HAs demonstrate better performance in Topology 3.
- When the attacking red agent is *red_meander*, all blue agents' performance deteriorates in Topology 3.

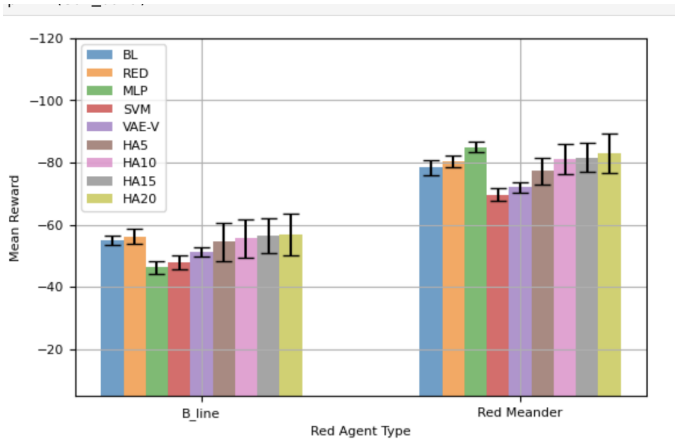

Figure 13. Performance Comparison Using Network Topology 3 - 100-Step Episode

Figure 14 shows the mean reward of the blue agents against both red agents over a 50-step episode. Generally, the trends in Figure 14 are similar to those in Figure 13. The agents trained using the proposed framework exhibit better mean rewards, with instances of statistically significantly better performance compared to a blue agent specifically trained against a particular red agent. Similarly, the trends in Table 12 mostly mirror those in Table 12, where SVM demonstrates the highest total mean reward, followed by VAE-V and MLP. Variants of HAs show the lowest total mean reward when considering their performance against both red agents. Comparing Figure 14 with Figure 8 reveals

 doi:10.20944/preprints202503.1421.v1

that, by and large, the trained blue agents demonstrate similar performance for both topologies. Also, there are instances where agents trained using the proposed framework demonstrate higher mean reward, i.e., better performance.

Table 12. Total Mean Reward - 50-Step Episode (Topology 3)

Agent Type	BL	RED	MLP	SVM	VAE-V
Total Mean Reward	-55.49	-54.6	-50.83	**-48.77**	-51.24
Agent Type	HA5	HA10	HA15	HA20	
Total Mean Reward	-57.98	-59.45	-60.4	-61.7	

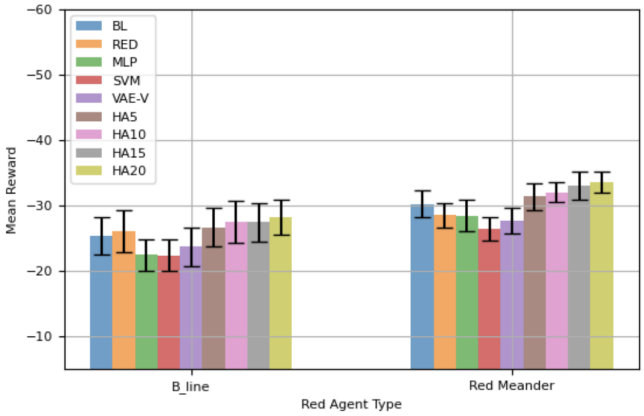

Figure 14. Performance Comparison Using Network Topology 3 - 50-Step Episode

In general, our single blue agents perform comparable to blue agents specifically trained against particular red agents, if not superior to them. This indicates that the proposed framework effectively develops a generic blue agent with regard to specific attackers. Since network topologies can vary from one network to another, it is crucial for a blue agent to adapt its behavior as the network topology changes. Therefore, we evaluated blue agents on network topologies different from those used for training (the total number of nodes in the network remained the same, but the number of nodes in each subnet was altered). Across these different network topologies, the blue agents trained with the proposed framework demonstrated statistically significantly better performance compared to other evaluated blue agents. This underscores the generalizability of the proposed framework in adapting to specific network topology changes.

5. Conclusion

The objective of our work is to train a single, generic blue agent that is independent of any attacking red agent, and to some extent, independent of network topology as well. The proposed framework combines RL with supervised learning. A variant of the framework uses a custom-built VAE in the supervised learning phase of the framework. To explore alternate methods that may result in a generic blue agent, the option of retraining an existing blue agent against another type of red agent is also explored. Our results show that the proposed framework for generic blue agent training does result in a blue agent that possess generic features in terms of an attacking red agent. Retraining an existing blue agent against another type of red agent tends to result in poor performance. The VAE-based version of the proposed framework demonstrated the best overall performance, i.e., achieved higher total rewards. It also demonstrated strong performance when evaluated in scenarios with network topologies that differed from the one used during agent training. Also, all studied supervised learning methods performed strongly, compared to blue agents trained solely using reinforcement learning. In fact, in one scenario (Topology 3), the use of a SVM outperformed the VAE-based version.

 doi:10.20944/preprints202503.1421.v1

SVM is also more effective in scenarios where the network topology was changed, compared to MLP. This indicates that using SVM in the supervised learning phase of the proposed framework may result in a blue agent that is more topology-agnostic.

For future work, we will explore the most suitable approach to achieve generalized strong performance in more depth. In particular, we will apply our approach to the recently launched Cage Challenge 4 [30]. This challenge has a number of interesting features. During both training and evaluation, each time the environment resets a new scenario is created. The overall network topology in terms of networks, subnets, and physcial connectivity remains the some. However, the number of user hosts and servers in each subnet varies, in addition to the number of deployed services on each user host and server. Also, the network experiences multiple operational phases. Different phases support different communication patterns based on pre-configured firewall rules. The reward function varies based on a device's priority during a given phase. Last but not least, Cage Challenge 4 models a multi-agent environment, where collectively 5 blue agents can be trained to defend the complete network, which differs significantly from the single defender scenario we studied here.

References

1. Farooq, M.O.; Wheelock, I.; Pesch, D. IoT-Connect: An Interoperability Framework for Smart Home Communication Protocols. *IEEE Consumer Electronics Magazine* **2020**, *9*, 22–29.
2. Yeong, D.J.; Velasco-Hernandez, G.; Barry, J.; Walsh, J. Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. *Sensors* **2021**, *21*.
3. Zhang, C.; Si, X.; Zhu, X.; Zhang, Y. A Survey on the Security of the Metaverse. In Proceedings of the 2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom), 2023, pp. 428–432.
4. Nichols, W.; Hill, Z.; Hawrylak, P.; Hale, J.; Papa, M. Automatic Generation of Attack Scripts from Attack Graphs. In Proceedings of the 2018 1st International Conference on Data Intelligence and Security (ICDIS), 2018, pp. 267–274.
5. Sultana, M.; Taylor, A.; Li, L. Autonomous network cyber offence strategy through deep reinforcement learning. In Proceedings of the Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III. International Society for Optics and Photonics, SPIE, 2021, Vol. 11746, p. 1174622.
6. Haque, N.I.; Shahriar, M.H.; Dastgir, M.G.; Debnath, A.; Parvez, I.; Sarwat, A.; Rahman, M.A. A Survey of Machine Learning-based Cyber-physical Attack Generation, Detection, and Mitigation in Smart-Grid. In Proceedings of the 52^{nd} North American Power Symposium (NAPS), 2021, pp. 1–6.
7. Ghanem, M.C.; Chen, T.M. Reinforcement Learning for Efficient Network Penetration Testing. *Information* **2020**, *11*.
8. Pozdniakov, K.; Alonso, E.; Stankovic, V.; Tam, K.; Jones, K. Smart Security Audit: Reinforcement Learning with a Deep Neural Network Approximator. In Proceedings of the 2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), 2020, pp. 1–8.
9. Fang, Z.; Wang, J.; Li, B.; Wu, S.; Zhou, Y.; Huang, H. Evading Anti-Malware Engines With Deep Reinforcement Learning. *IEEE Access* **2019**, *7*, 48867–48879.
10. Pan, Z.; Sheldon, J.; Mishra, P. Hardware-Assisted Malware Detection using Explainable Machine Learning. In Proceedings of the 2020 IEEE 38th International Conference on Computer Design (ICCD), 2020, pp. 663–666.
11. Han, G.; Xiao, L.; Poor, H.V. Two-dimensional anti-jamming communication based on deep reinforcement learning. In Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 2087–2091.
12. Pu, Z.; Niu, Y.; Zhang, G. A Multi-Parameter Intelligent Communication Anti-Jamming Method Based on Three-Dimensional Q-Learning. In Proceedings of the 2022 IEEE 2nd International Conference on Computer Communication and Artificial Intelligence (CCAI), 2022, pp. 205–210.
13. Huang, M. Theory and Implementation of linear regression. In Proceedings of the 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), 2020, pp. 210–217.
14. Zou, X.; Hu, Y.; Tian, Z.; Shen, K. Logistic Regression Model Optimization and Case Analysis. In Proceedings of the 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 2019, pp. 135–139.

 doi:10.20944/preprints202503.1421.v1

15. Hearst, M.; Dumais, S.; Osuna, E.; Platt, J.; Scholkopf, B. Support vector machines. *IEEE Intelligent Systems and their Applications* **1998**, *13*, 18–28.
16. Louppe, G. Understanding Random Forests: From Theory to Practice, 2015, [arXiv:stat.ML/1407.7502].
17. Mundt, M.; Hong, Y.; Pliushch, I.; Ramesh, V. A Wholistic View of Continual Learning with Deep Neural Networks: Forgotten Lessons and the Bridge to Active and Open World Learning. *Neural Networks* **2023**, *160*, 306–336.
18. Naeem, M.; Rizvi, S.T.H.; Coronato, A. A Gentle Introduction to Reinforcement Learning and its Application in Different Fields. *IEEE Access* **2020**, *8*, 209320–209344.
19. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. *Nature* **2015**, *518*, 529–533. https://doi.org/10.1038/nature14236.
20. Molina-Markham, A.; Miniter, C.; Powell, B.; Ridley, A. Network Environment Design for Autonomous Cyberdefense, 2021, [arXiv:cs.CR/2103.07583].
21. Li, L.; Rami, J.P.S.E.; Taylor, A.; Rao, J.H.; Kunz, T. Unified Emulation-Simulation Training Environment for Autonomous Cyber Agents, 2023, [arXiv:cs.LG/2304.01244].
22. Li, L.; Fayad, R.; Taylor, A. CyGIL: A Cyber Gym for Training Autonomous Agents over Emulated Network Systems, 2021, [arXiv:cs.CR/2109.03331].
23. Cyber Battle Sim. https://github.com/microsoft/CyberBattleSim. Last accessed: 20th Jan, 2024.
24. Baillie, C.; Standen, M.; Schwartz, J.; Docking, M.; Bowman, D.; Kim, J. CybORG: An Autonomous Cyber Operations Research Gym, 2020, [arXiv:cs.CR/2002.10667].
25. OpenAI Gym. (https://gymnasium.farama.org). "Last accessed: 5th August, 2024.
26. Kiely, M.; Bowman, D.; Standen, M.; Moir, C. On Autonomous Agents in a Cyber Defence Environment, 2023, [arXiv:cs.CR/2309.07388].
27. Kunz, T.; Fisher, C.; Novara-Gsell, J.L.; Nguyen, C.; Li, L. A Multiagent CyberBattleSim for RL Cyber Operation Agents, 2023, [arXiv:cs.CR/2304.11052].
28. Foley, M.; Hicks, C.; Highnam, K.; Mavroudis, V. Autonomous Network Defence Using Reinforcement Learning, New York, NY, USA, 2022; ASIA CCS '22, p. 1252–1254.
29. Optuna: A hyperparameter optimization framework. (https://optuna.readthedocs.io/en/stable/). "Last accessed: 7th May, 2024.
30. TTCP CAGE Working Group. TTCP CAGE Challenge 4. https://github.com/cage-challenge/cage-challenge-4, 2023.