A Generic Blue Agent Training Framework for Autonomous Cyber Operations

Muhammad Omer Farooq[1] and Thomas Kunz[2]
[1]*Department of Electronics and Computer Engineering, University of Limerick, Ireland*
[2]*Department of Systems and Computer Engineering, Carleton University, Canada*
omer.farooq@ymail.com, tkunz@sce.carleton.ca

***Abstract*—Sophisticated mechanisms for attacking a computer network are emerging, therefore it is of great importance that equally sophisticated mechanisms should be in place to defend against malicious attacks on the network. Autonomous cyber operations (ACO) is considered to be a potential option to provide timely defense against malicious attacks. In ACO, an agent that tries to attack a network is referred to as red agent, and an agent that defends against the red agent is called blue agent. In real scenarios, different kinds of red agents can attack a network, hence a blue agent needs to defend against a variety of red agents, each with their own attack strategy and specific goal. However, it is a challenging task to train a blue agent that is agnostic of the red agent. Hence, we present here a framework for generic blue agent training, i.e., training a blue agent that can defend against different kinds of red agents. The framework is a combination of reinforcement learning and supervised learning. Our results demonstrate that the presented framework for generic blue agent training does exhibit generic characteristics, and the framework does demonstrate better performance compared to an alternate approach.**

***Index Terms*—Autonomous Cyber Operations (ACOs), Cyber Security, Blue Agent, Red Agent, Machine Learning.**

I. Introduction

The increase in digital connectivity with every passing day is rapidly changing our society in a number of ways. At the core of digital connectivity is a connected network of computer systems. Such computer systems do not only provide connectivity, but also offer storage and computational resources. We are at a stage where a disruption in our digital connectivity and operations can have a substantial negative impact. Hence, securing connected computer systems is of immense importance. Recently, attackers are using machine learning (ML) as a tool to attack a networked system [1]. This poses a significant challenge to the system's security, and only human-based response may not be sufficient as possibly such a response may require a relatively long time to detect the attack. Hence, there is a need to develop automated intelligent methods to timely detect and respond to attacks. In this regard, autonomous cyber operations (ACO) deal with strengthening security of the computer system or a set of computer systems through machine-based decision making. ACO provides the following benefits: timely attack detection, protection of isolated systems, and protection of systems where there is a lack of trained human resources. In ACO, two types of agents are considered: blue and red. A blue agent is responsible for defending against malicious attacks, whereas a red agent tries to carry out malicious activities over the system.

ISBN 978-3-903176-63-8 ©2024 IFIP

Use of reinforcement learning (RL) has been explored in many avenues of computer networks' security, such as, penetration testing [2] malware detection [3], anti-jamming communication system [4], etc. The existing work demonstrates that in networked systems' security RL can play a crucial role. In RL for ACO, a computer network environment along with blue and red agents is modelled as a game with defined actions for red and blue agents, states for the network, and rewards. The abstracted computer network environment is used by the agents to learn an effective decision making process. A computer network can consists of a large number of different types of entities, and is dynamic, hence modelling such a system for RL requires a large state space. In addition, agents have at their disposal a large variety of possible actions, resulting in a (potentially) large action space. Large state and action spaces make training an effective agent for ACO a challenging task.

Here, a combination of RL and supervised learning is explored to train a generic blue agent that can defend a system against different kinds of red agents. Specifically, training a blue agent to defend well against a particular type of red agent is not a practical solution as it requires deploying multiple blue agents, one per potentially deployed red agent. In a scenario where multiple blue agents are deployed, a blue agent's actions may conflict with another blue agent's action, hence the requirement for a single generic blue agent. The following are main contributions of this research paper:

- A framework for generic blue agent training that can deal with different kinds of red agents.
- Highlighting the role of HP optimization in training a generic blue agent.
- Highlighting limitations of retraining blue agents against a different kind of red agent in an attempt to come up with a generalised blue agent.
- Experimental results that highlight effectiveness of the proposed framework.

The rest of this paper as organised as follows. Background is discussed in Section II. The details of the proposed framework for a generic blue agent training are provided in Section III. Performance evaluations are presented in Section IV, and finally conclusions and future work are discussed in Section V.

II. Background

A. Reinforcement Learning

RL is a ML paradigm [5] in which an agent/machine trains itself by interacting with an environment. 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 for training, but it requires an environment that is consistent with the target domain. Defining RL states and actions needs domain knowledge, and the level of accuracy in defining these impacts the resulting agent quality. RL seems to be a promising choice for ACO as red or blue agents can learn optimal sequence of actions by interacting with the training environment.

B. Reinforcement Learning Environments for Autonomous Cyber Operations

In [6], an important point is highlighted, i.e., a computer network defense cannot be modelled as a single game for RL as there is a dynamic spectrum of network configurations and attacks. Based on this fact, a RL framework for autonomous network defense is presented. The framework enables creation of multiple network environments to apply RL for autonomous network defense. In [7], RL environments for cyber operations are presented. High-fidelity training takes place in an emulation environment. Emulation-based training also provides information to construct a simulator that can latter be used to train red and blue agents. In [8], Cyber Gym for Intelligent Learning (CyGIL) is presented. CyGIL cannot only run in an emulated environment, but it can also work over a real network. CyGIL comes with tools necessary for red team activities, and it also incorporates a comprehensive action space for a red agent training. Moreover, it comes with various games and objectives to train red agents for different types of attacks.

'CyberBattleSim' is an open source cyber-battle simulation tool launched by Microsoft [9]. It allows for the creation of cyber defense environments, and it comes with RL capabilities and a red agent. CyberBattleSim requires significant human effort to define a successful exploit, and an attempt to overcome this limitation is made in [10] by integrating an external knowledge source with the simulator. The source contains semantic information about network entitles, and also contains information that helps agents to reach their goals in fewer steps. Moreover, red and blue agents can be trained simultaneously. Similarly, in [11] CyberBattleSim is extended to train blue agents, moreover it enables simultaneous training of red and blue agents in the simulator.

CybORG [12] provides a set of tools and networking environments to train blue agents for ACO using RL. CybORG supports training in simulation and emulation-based environments. A typical learning environment in CybORG consists of a computer network that is further partitioned into different subnets. A 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 Fig. 1. CybORG comes with red and blue agents. A red agent tries to attack a network and disrupt its operation, whereas a blue agent tries to defend the network against the red agent. There are two types of red agents in CybORG: $(i) red\,meander$ and $(ii)\, b\_line$. b_line is called baseline agent. Furthermore, CybORG also comes with a green agent whose task is to generate a network's users normal traffic.

C. Reinforcement Learning Based Blue Agents' Training Methods

Most of the existing RL-based blue agents for ACOs can be categorised as [13]: single agent, hierarchical agent, and ensembles.

Single agents are simple as only one agent is trained using RL to defend against different types of red agents, and only one blue agent is deployed in a networked system [11]. Hierarchical agents are relatively complex as they usually come with an agent that is used to select another agent to provide an action based on the type of red agent attacking a networked system [14]. Similarly, ensemble approaches train a number of agents, and each agent receives a network's state, and based on the state the agents provide an action. The most commonly reported action is finally executed. A number of existing blue agents based on an ensemble approach are discussed in [13]. It has been discussed in [13] that in most of the scenarios hierarchical agents demonstrate best performance, followed by ensembles, and single agent.

D. Discussion

To support ACO, training generalized blue agent is a challenging task, but due to its importance a number of efforts have been made to design an RL method that can yield a generalized blue agent. On this front hierarchical RL methods have demonstrated better performance compared to ensembles-based RL and single-agent RL methods. However, hierarchical methods not only require training of multiple agents, but they also require deployment of multiple blue agents in a networked system. The latter point is also true for ensembles-based RL methods. This makes these approaches relatively more resource intensive and real-time decision making can take a relatively long as multiple agents are involved in the process. Hence, there is a need to further explore possibilities of training a blue agent that does not only possess generic features, but requires only one blue agent to be deployed in the networked system.

III. Framework for Generic Blue Agent Training

To support ACO, here details of the proposed framework are presented. Among other things, the framework combines RL and supervised machine learning to train a generalised blue agent, i.e., an agent that can defend against different types of red agents. Figure 2 shows the framework for

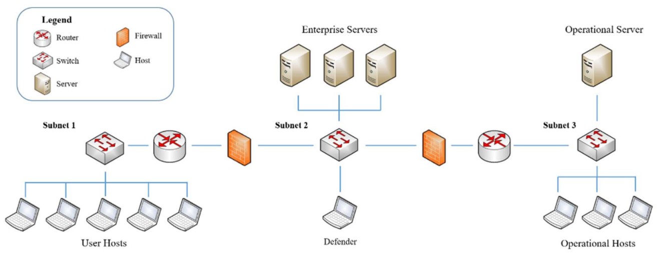

Fig. 1. A Networking Scenario in CybORG [12]

a generalised blue agent training. The components of the framework are discussed below.

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

In the proposed framework, initially a separate blue agent needs to be trained against each possible red agent, i.e., if there are N red agents in a system, there is a need to train N blue agents; one blue agent for a particular red agent. In ML, it is well-known that tuning possible values for HPs can improve a trained model's performance. For HP tuning a range of values are explored corresponding to different HPs associated with a given RL algorithm. The following strategy is used for HP tuning.

- Create a RL-based learning environment that among other entities (host, servers, firewalls, etc.) also includes a blue agent and a corresponding red agent.
- Supply a range of values for all possible HPs.
- Use a HP tuning library for RL, such as Optuna [15], to find the best HPs for a blue agent training.
- Store the optimal HPs' values obtained in the last step.
- Repeat the above steps for each blue agent that needs to be trained.

B. Reinforcement-Learning-Based Blue Agent Training

After searching optimal HPs for each blue agent, the agents need to be trained against their respective red agents. Blue agent training requires the creation of a learning environment consisting of a real network scenario. The RL algorithm for a blue agent's training must use the searched optimal HPs for the blue agent against a particular red agent. The blue agent is then trained for a large number of RL training steps against the red agent. The described process needs to be repeated for each blue agent's training.

C. Data Collection - Observations and Action Tuples

The proposed framework also uses supervised learning to train a generic blue agent for ACO. Supervised learning requires data for training, therefore the framework comes with a data collection mechanism. The framework trains a separate blue agent for each red agent using RL, therefore during training a blue agent learns to take best possible action based on current network state against the red agent it was trained against. Similarly, a generic blue agent also needs to learn best possible actions to take based on different possible network states, and from the state information the generic blue agent also needs to learn how to identify the type of red agent attacking the network.

To train a generic blue agent, the framework proposes to collect a sufficient number of observation-action tuples by running a scenario where a trained blue agent defends against the particular red agent, i.e., the red agent against whom the blue agent was originally trained. This needs to be carried out for all blue agents that were trained as described in Section III-B. The motivation here is that by collecting observation-action tuples corresponding to each trained blue agent in the system, the framework will have sufficient data that can be used to train a generic blue agent that will be capable of identifying the type of red agent attacking a network, and can learn to take appropriate actions. Algorithm 1 shows the data collection process.

D. Randomizing Sequencing of Collected Data

The data collection module collects data in a sequential manner, i.e., each blue agent is tried against a red agent against whom the agent was trained to collect observations-action tuples. To prepare data for supervised learning, the collected data needs to be consolidated in a single source file. After storing the data in the file, the sequencing of data needs to be randomised as otherwise records corresponding to each blue agent appear in sequence. This may impact the performance of a supervised ML algorithm. For example, while training there are chances that a neural network may not train too well corresponding to blue agents whose data was at the start of the file. Figure 3 shows the process of preparing data to train a generic blue agent.

E. Supervised Learning - Hyper-parameters Optimization

Once data is ready for training a generic blue agent using supervised learning, the next step in the process is to use the data and find the optimal HPs values for a suitable supervised ML algorithms, for example, multi-layer preceptors (MLP), support vector machine (SVM), random forest, etc. Based on the selection of a specific supervised learning algorithm, different number and types of HPs need to be tuned, for example, a MLP needs to tune the number of hidden layers, activation algorithm, learning rate, etc. Existing machine learning libraries, such as SciKit learn, come with application programming interfaces for HP tuning. The following steps summarises the process to search the optimal HPs' values:

- Select a suitable supervised ML algorithm.
- Select a reasonable range of values for different HPs associated with the selected algorithm.
- Obtain the best HPs' values using HP optimization library, such as SciKit learn.

F. Training a Generic Blue Agent

Most of the ingredients required to train a generic blue agent using a supervised ML algorithm are available by now thanks to the preceding modules of the framework. However, data for testing the trained blue agent is needed. For this purpose as typically done in ML, the framework proposes to partition the collected data into training and testing sets. Typically, 80% of collected data is used for training, and

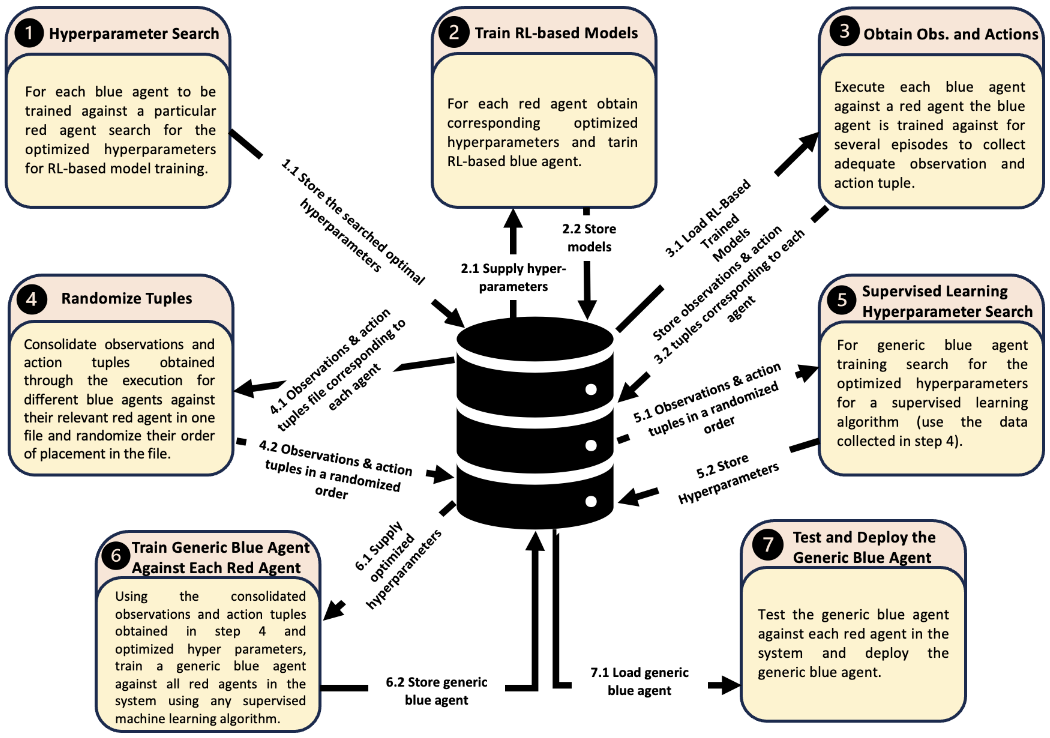
1 Hyperparameter Search
For each blue agent to be trained against a particular red agent search for the optimized hyperparameters for RL-based model training.
2 Train RL-based Models
For each red agent obtain corresponding optimized hyperparameters and tarin RL-based blue agent.
3 Obtain Obs. and Actions
Execute each blue agent against a red agent the blue agent is trained against for several episodes to collect adequate observation and action tuple.
4 Randomize Tuples
Consolidate observations and action tuples obtained through the execution for different blue agents against their relevant red agent in one file and randomize their order of placement in the file.
5 Supervised Learning Hyperparameter Search
For generic blue agent training search for the optimized hyperparameters for a supervised learning algorithm (use the data collected in step 4).
6 Train Generic Blue Agent Against Each Red Agent
Using the consolidated observations and action tuples obtained in step 4 and optimized hyper parameters, train a generic blue agent against all red agents in the system using any supervised machine learning algorithm.
7 Test and Deploy the Generic Blue Agent
Test the generic blue agent against each red agent in the system and deploy the generic blue agent.
1.1 Store the searched optimal hyperparameters
2.1 Supply hyper-parameters
2.2 Store models
3.1 Load RL-Based Trained Models
Store observations & action
3.2 tuples corresponding to each agent
4.1 Observations & action tuples file corresponding to each agent
4.2 Observations & action tuples in a randomized order
5.1 Observations & action tuples in a randomized order
5.2 Store Hyperparameters
6.1 Supply optimized hyperparameters
6.2 Store generic blue agent
7.1 Load generic blue agent

Fig. 2. Proposed Framework for Generalised Blue Agent

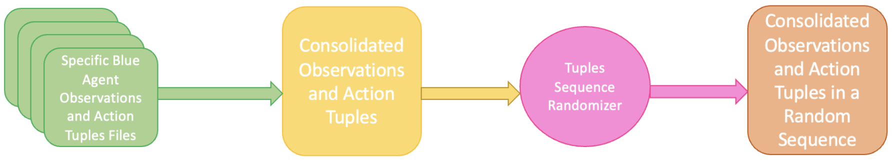
Specific Blue Agent Observations and Action Tuples Files
Consolidated Observations and Action Tuples
Tuples Sequence Randomizer
Consolidated Observations and Action Tuples in a Random Sequence

Fig. 3. Preparing Data for Supervised Learning

20% is used for testing. As the framework collects data using blue agents trained through RL, therefore a large amount of training and testing data can be collected. Using the training data set and the optimised HPs' values a generic blue agent is trained through a supervised ML algorithm.

G. *Test and Deploy Generic Blue Agent*

After training the generic blue agent, the agent needs to be tested using the test data set. If after testing the agent's performance is acceptable, the agent can be deployed on a real-system. On the real system periodically the state of the system is fed to the agent, and the agent suggests an appropriate action based on the provided system state. The action is provided to a script that may executes the action.

IV. Performance Evaluation

We deployed our approach in CybORG [12], which provides two distrinct red agents/attackers, training a single defensive blue agent. Initially, for performance evaluation a separate blue agent is trained against $red\, meander$ and b_line red agents. Each blue agent is trained for $100,000$ steps. To investigate the impact of HP optimization on the performance of blue agents, we determined the optimal values for HPs for a blue agent training against the $red\, meander$ and b_line red agents. For the different performance evaluation scenarios considered here, Proximal Policy Optimization (PPO) is used as the RL algorithm, as previous studies demonstrated its superior performance and generalised behaviour compared to other RL algorithms, such as DQN [16]. For performance evaluation, the networking scenario shown in Figure 1 is used.

A. *Hyper-Parameter Optimization for Blue Agents*

For HP optimization, the Optuna [15] package has been used. The PPO algorithm specifically, and RL in general, has many HPs that can be tuned. Table I list the HPs that have been tuned and the range of values considered. For each combination of HP values a blue agent is trained against the red agent of interest for $50,000$ steps, and the model is evaluated after every $5,000$ steps. During evaluation the model is evaluated for a couple of episodes, each episode consisting of 100 steps. The values for HPs that give the best reward during the evaluation are the values of choice for the HPs. The best blue agent reward against the b_line and $red\, meander$ agents are -33.30 and -53.20 respectively. Table II shows the tuned HP values for the blue agents.

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
$return\ list\_of\_obs\_actions\_files$;
```

TABLE I
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])]

B. Impact of Hyper-Parameter Tuning on a Blue Agent

To analyse the impact of HP tuning on a blue agent's performance, a separate blue agent is trained against each red agent, i.e., b_line and $red\,meander$. One set of blue agents was trained without HP tuning, and another was trained with HP tuning. For training without HP tuning, the default values for HPs available in the Stable-Baselines3 RL library are used. Moreover, to analyse generalizeability of a trained blue agent, the blue agent trained against b_line red agent is also evaluated against the $red\,meander$ agent and vice versa. A blue agent is evaluated for 100 episodes, and each episode consists of 100 steps. The performance of an agent is evaluated using the total reward metric per episode. Fig. 4 shows the blue agents' performance in the form of box and whisker plots. The description of the labels used along x-axis in plots is given in Table III. The following are key takeaways from Fig. 4:

- Training blue agents with HP optimization significantly improves the blue agents' performance.
- Improvement in the performance of blue agents when the agents were trained by incorporating HP optimization is agnostic of the red agent against whom a blue agent is evaluated.
- The performance demonstrated by RED-BL-HP is similar to the performance demonstrated by BL-HP. BL-HP demonstrates slightly higher mean and mostly better total reward in the 4^{th} quartile. However, BL-RED-HP demonstrates poor performance compared to RED-HP.

TABLE II
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]

C. Retrained Blue Agents

In the previous section, we have seen that a blue agent that is trained against the b_line red agent has poor performance when the agent was evaluated against the $red\,meander$ agent. Similarly, a blue agent that was trained against the $red\,meander$ agent, when evaluated against the b_line red agent, performs not quite as good as when the baseline blue agent was evaluate against the b_line red agent. Hence, the trained blue agents do not demonstrate generalizability, i.e., a blue agent trained against one red agent does not demonstrate the same-level of performance against the other red agent. The previous statement holds strongly for the baseline blue agent. To explore ways to train a single blue agent with strong generalizability, the existing blue agents are retrained in the following manner:

- An existing blue agent that was trained against b_line red agent is retrained against the $red\,meander$ agent. Hereafter, this agent is referred as BL-RT-RED.
- An existing blue agent that was trained against the $red\,meander$ agent is retrained against the b_line red agent. Hereafter, this agent is referred as RED-RT-BL.

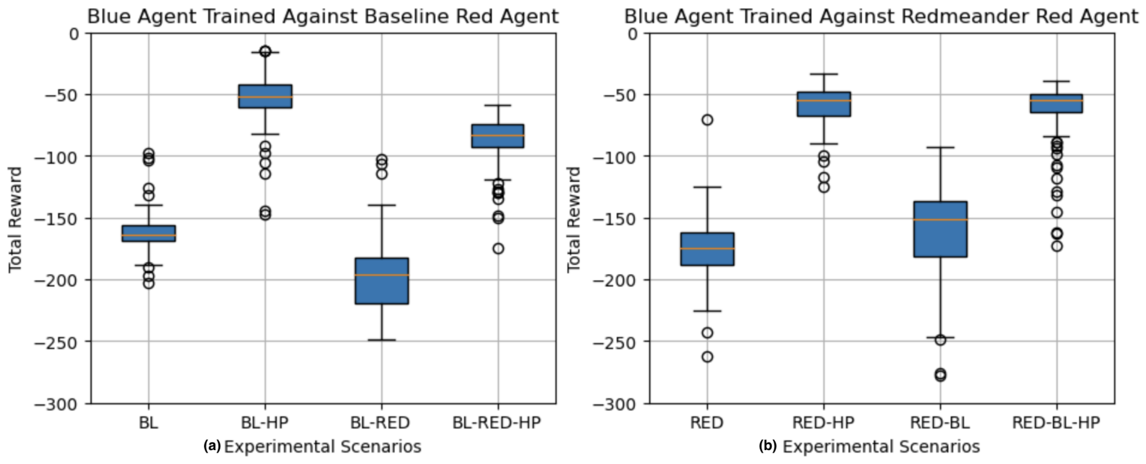

Fig. 4. Blue Agents' Performance Comparison

TABLE III
NOMENCLATURE RELATED TO EXPERIMENTAL RESULTS

Label	Description
BL	Blue agent trained against the b_line red agent without HP optimization.
BL-HP	Blue agent trained against the b_line red agent with HP optimization.
BL-RED	Blue agent trained against the b_line red agent without HP optimization, but evaluated against the $red\,meander$ agent.
BL-RED-HP	Blue agent trained against the b_line red agent with HP optimization, but evaluated against the $red\,meander$ agent.
RED	Blue agent trained against the $red\,meander$ agent without HP optimization.
RED-HP	Blue agent trained against the $red\,meander$ agent with HP optimization.
RED-BL	Blue agent trained against the $red\,meander$ agent without HP optimization, but evaluated against the b_line red agent.
RED-BL-HP	Blue agent trained against the $red\,meander$ agent with HP optimization, but evaluated against the b_line red agent.
BL-RT-RED	Existing blue agent for the b_line red agent retrained against the $red\,meander$ agent.
RED-RT-BL	Existing blue agent for the $red\,meadner$ red agent retrained against the b_line red agent.
MLP-HP	MLP-based agent with HP optimization (based on the proposed framework).
SVM-HP	SVM-based agent with HP optimization (based on the proposed framework).

Each agent is retrained for another $100,000$ steps using the optimal HP values for the relevant red agent; i.e., the agent against whom the retraining took place. The performance achieved with these retrained agents will be compared against the generic agent we trained using the proposed framework.

D. Proposed Framework Setup

The proposed framework is a combination of RL and supervised learning. To collect data for supervised learning, each trained blue agent is executed against the red agent it was trained against and observation and action tuples are collected for supervised learning. For each agent $10,000$ tuples were collected. We use multi-layer perceptron (MLP) and support vector machine (SVM) for supervised learning, but the proposed framework can be used with any supervised learning algorithm. HP optimization has been performed for both MLP and SVM using Bayes search. For SVM, C, $gamma$, and $kernel$ HPs are considered, and the optimal searched values for these parameters using the collected data are 10.9975, $4.501669e-06$, and $linear$ respectively. Similarly, for MLP, hidden layers, activation function, solver, learning rate, and alpha HPs are considered, and the optimal values determined for these parameters using the collected data are 10, $tanh$, $lbfgs$, $invscaling$, and 0.01542674 respectively. The resulting generic agents are called MLP-HP and SVM-HP.

E. Agents' Performance Comparison

In total, we have six blue agents that are evaluated for their generalizability: BL-HP, RED-HP, BL-RT-RED, RED-RT-BL, MLP-HP, and SVM-HP. Table IV summarizes their performance. The metrics reported in Table IV are related to the total reward: mean total rewards (mean), standard deviation (Std), and an agent's mean total reward against both red agents (Total Reward).

Among other agents Table IV also shows the total mean reward demonstrated by retrained blue agents (BL-RT-RED and RED-RT-BL) along with related metrics. Comparison of the BL-RT-RED agent with BL-HP reveals the following: (i) performance of the retrained agent against the b_line agent has deteriorated, (ii) the agent demonstrates a somehwat improved performance against the $red\,meander$ agent, and (iii) the agent's performance against the $red\,meander$ agent is poor compared to an agent that was only trained against $red\,meander$ (RED-HP). Similarly, a comparison of RED-RT-BL with RED-HP reveals the following about the blue agent that was originally trained against $red\,meander$: (i) RED-RT-BL performance against the $red\,meander$ agent has deteriorated, and (ii) the agent's performance against the b_line agent has improved a little. Moreover, both retrained blue agents demonstrate lowest $Total\ Reward$ compared to other blue agents. The above highlights that retraining the existing agents against another agent does not introduce generalizability, and in fact it is counter-productive. The counter-productivity is due to the fact that when an existing

TABLE IV
EXPERIMENTAL RESULTS - 100-STEP EPISODE

Agent Name	**b_line**		**Red Meander**		**Total Reward**
	Mean	**Std**	**Mean**	**Std**	
BL-HP	-53.68	22.41	-87.20	21.1	-140.88
RED-HP	-64.30	26.80	-58.66	17	-122.96
BL-RT-RED	-67.85	29.13	-83.48	15.38	-155.33
RED-RT-BL	-61.55	24.87	-137.20	66.10	-198.75
MLP-HP	-55.20	25.40	-62.50	18	**-117.7**
SVM-HP	-56.20	26.60	-68.41	24.17	-124.61

TABLE V
EXPERIMENTAL RESULTS - 50-STEP EPISODE

Agent Name	**b_line**		**Red Meander**		**Total Reward**
	Mean	**Std**	**Mean**	**Std**	
BL-HP	-23	11.4	-34.4	9.6	-57.4
RED-HP	-32.86	14	-25.97	6.9	-58.83
MLP-HP	-23.85	11.80	-28.54	7.75	**-52.39**
SVM-HP	-23.92	12	-29	10.45	**-52.92**

agent is retrained against a new agent, over the period of training the agent slowly forgets the learned features of the previous agent against whom the agent was trained.

Table IV also shows the total mean reward along with related metrics as demonstrated by the agents trained through the proposed framework (MLP-HP and SV-HP). Comparing the generic MLP-HP and SVM-HP agents with BL-HP reveals that MLP-HP and SVM-HP perform similar to BL-HP against the b_line red agent. However, MLP-HP and SVM-HP outperform BL-HP when the agents were evaluated against the $red\,meander$ agent. Comparing the two generic agents against RED-HP reveals that MLP-HP and SVM-HP performance against $red\,meander$ is not too far off from the mean total reward demonstrated by RED-HP. RED-HP demonstrates inferior performance when evaluated against the b_line agent compared to the performance demonstrated by MLP-HP and SVM-HP against the b_line agent. MLP-HP and SVM-HP demonstrate comparable standard deviation when compared to BL-HP and RED-HP. Moreover, MLP-HP demonstrates best $Total\,Reward$ amongst all the evaluated blue agents. Hence, overall the proposed generic agent exhibits the strongest generalizability and acceptable level of performance relatively speaking.

F. Results Based on Shorter Episodes

Previously, all results are based on an episode length of 100 steps. Here, results corresponding to another set of experiments are presented, based on an episode length of 50 steps. Previous results have demonstrated that retraining of agents is counter-productive, hence here retrained agents are not considered. Table V shows the mean total rewards along with related metrics as demonstrated by different blue agents. The performance comparison of different agents demonstrates that MLP-HP and SVM-HP demonstrate better performance against both red agents compared to the blue agents that were trained against one type of red agent, but evaluated against the other type of red agent. Moreover, the performance demonstrated by MLP-HP and SVM-HP against both types of red agents is not to far off in comparison to the performance demonstrated by the blue agents against the red agents they were trained against. The trends demonstrated here are consistent with 100 steps episode results, but in this case SVM-HP also demonstrates better $Total\,Reward$ compared to BL-HP and RED-HP.

V. CONCLUSIONS AND FUTURE WORK

A generic blue agent training framework for ACO is presented with the aim to train a blue agent that is agnostic of an attacking red agent. The impact of HP optimization on the performance of a blue agent is analysed. Our results show that the framework for generic blue agent training does result in a blue agent that possess generic features. Furthermore, it has been demonstrated that the HP optimizing has a positive impact on a trained blue agents' performance. In the future, we plan to explore the impact of suitable manifold learning methods on the training and performance of a generic blue agent. Furthermore, we also plan to explore the impact of other existing supervising learning methods, such as decision trees, random forest, XGBoost, etc. on the performance of a trained generic blue model.

REFERENCES

[1] N. I. Haque, M. H. Shahriar, M. G. Dastgir, A. Debnath, I. Parvez, A. Sarwat, and M. A. Rahman, "A Survey of Machine Learning-based Cyber-physical Attack Generation, Detection, and Mitigation in Smart-Grid," in 52^{nd} *North American Power Symposium (NAPS)*, 2021, pp. 1–6.

[2] M. C. Ghanem and T. M. Chen, "Reinforcement Learning for Efficient Network Penetration Testing," *Information*, vol. 11, no. 1, 2020.

[3] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, and H. Huang, "Evading Anti-Malware Engines With Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 48 867–48 879, 2019.

[4] Z. Pu, Y. Niu, and G. Zhang, "A Multi-Parameter Intelligent Communication Anti-Jamming Method Based on Three-Dimensional Q-Learning," in *2022 IEEE 2nd International Conference on Computer Communication and Artificial Intelligence (CCAI)*, 2022, pp. 205–210.

[5] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, 2020.

[6] A. Molina-Markham, C. Miniter, B. Powell, and A. Ridley, "Network Environment Design for Autonomous Cyberdefense," 2021.

[7] L. Li, J.-P. S. E. Rami, A. Taylor, J. H. Rao, and T. Kunz, "Unified Emulation-Simulation Training Environment for Autonomous Cyber Agents," 2023.

[8] L. Li, R. Fayad, and A. Taylor, "CyGIL: A Cyber Gym for Training Autonomous Agents over Emulated Network Systems," 2021.

[9] "Cyber Battle Sim," https://github.com/microsoft/CyberBattleSim, last accessed: 20^{th} Jan, 2024.

[10] A. Piplai, M. Anoruo, K. Fasaye, A. Joshi, T. Finin, and A. Ridley, "Knowledge Guided Two-player Reinforcement Learning for Cyber Attacks and Defenses," in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2022, pp. 1342–1349.

[11] T. Kunz, C. Fisher, J. L. Novara-Gsell, C. Nguyen, and L. Li, "A Multiagent CyberBattleSim for RL Cyber Operation Agents," 2023.

[12] C. Baillie, M. Standen, J. Schwartz, M. Docking, D. Bowman, and J. Kim, "CybORG: An Autonomous Cyber Operations Research Gym," 2020.

[13] M. Kiely, D. Bowman, M. Standen, and C. Moir, "On Autonomous Agents in a Cyber Defence Environment," 2023.

[14] M. Foley, C. Hicks, K. Highnam, and V. Mavroudis, "Autonomous Network Defence Using Reinforcement Learning," ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1252–1254.

[15] "Optuna: A hyperparameter optimization framework," (https://optuna.readthedocs.io/en/stable/), "Last accessed: 7^{th} Jan, 2024.

[16] M. Sultana, A. Taylor, and L. Li, "Autonomous network cyber offence strategy through deep reinforcement learning," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, vol. 11746, International Society for Optics and Photonics. SPIE, 2021, p. 1174622.