Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 33, Num. 4 (2026) 68-81

RESEARCH ARTICLE

Behavioral Insights of SARSA-Based Reinforcement Learning for IPv6 Intrusion Classification

Insights Comportamentais da Aprendizagem por Reforço Baseada em SARSA para Classificação de Invasões IPv6

April Firman Daru[1]*, Alauddin Maulana Hirzan[1], Zainab Senan Mahmod Attar Bashi[2], Fajriannoor Fanani[1]

Abstract: Intrusion Detection Systems (IDS) are crucial for network security, but traditional models like signature-based and supervised learning approaches require frequent updates or retraining. This study introduces a reinforcement learning-based IDS using SARSA for IPv6 intrusion detection. The SARSA agent with epsilon 0.1 (E-0.1) performed best: optimal detection at step 11, 193,453 correct and 25,014 incorrect actions, total reward 842,195, average reward 3.855 per epoch, outperforming E-0.5 and E-0.9. Furthermore, a comparison of Q-Table exploitation between Q-Learning and SARSA indicates that both algorithms yield similar results. However, SARSA is preferable in security contexts because its more realistic approach results in a lower false alarm rate. This behavioral distinction is demonstrated by a -1.2109 value difference and a substantially lower mean state value for SARSA (4.903) compared to the more optimistic Q-Learning (16.310). To ensure robustness, the results were validated using different random seeds, and E-0.1 consistently produced stable total rewards with minimal variance. Additionally, Shannon entropy confirmed the agent's stable decision-making behavior, where E-0.1 showed faster convergence, indicating efficient learning. Altogether, these findings suggest that the SARSA-based IDS offers a more adaptive and reliable solution for intrusion detection, particularly in dynamic IPv6 environments.
Keywords: epsilon greedy — IPv6 — intrusion classification — policy behavior analysis — SARSA algoritm

Resumo: Os Sistemas de Detecção de Intrusão (IDS) são essenciais para a segurança de redes, mas métodos tradicionais, como abordagens baseadas em assinaturas e aprendizado supervisionado, exigem atualizações frequentes e retreinamento. Este estudo propõe um IDS baseado em aprendizado por reforço utilizando o algoritmo SARSA para detecção de intrusões em ambientes IPv6. O agente SARSA com epsilon 0,1 (E-0,1) apresentou o melhor desempenho, alcançando detecção ideal na etapa 11, com 193.453 ações corretas, 25.014 incorretas, recompensa total de 842.195 e média de 3,855 por época, superando os agentes E-0,5 e E-0,9. A comparação entre Q-Learning e SARSA mostrou resultados semelhantes na exploração da Tabela Q. Contudo, o SARSA demonstrou menor taxa de falsos alarmes devido à sua abordagem mais realista, evidenciada pela diferença de valor de -1,2109 e pelo menor valor médio de estado (4,903) em relação ao Q-Learning (16,310). Os resultados também foram validados com diferentes sementes aleatórias, confirmando estabilidade e convergência mais rápida do E-0,1. Assim, o IDS baseado em SARSA apresenta-se como uma solução adaptável e confiável para ambientes IPv6 dinâmicos.
Palavras-Chave: algoritmo SARSA — análise de comportamento de políticas — classificação de invasões — epsilon greedy — IPv6

[1] *Faculty of Information Technology and Communication, Universitas Semarang (USM), Indonesia*
[2] *Kulliyyah of Information and Communication Technology, International Islamic University Malaysia (IIUM), Malaysia*
*__Corresponding author__: maulanahirzan@usm.ac.id
DOI: http://dx.doi.org/10.22456/2175-2745.148437 • **Received:** 27/06/2025 • **Accepted:** 02/07/2026

1. Introduction

An Intrusion Detection System, or IDS, is software capable of sniffing flowing network traffic and detecting anomalies inside a packet. Although IDS works similarly to antimalware scanning viruses or other malware, it only focuses on packet data within network traffic. This software uses many methods

to determine whether a packet is an intrusion or a regular packet[1]. Using a rule set or signature database is the easiest method to detect an intrusion. This type is the most effective way to detect intrusion. Snort IDS is one of many examples of rule-based intrusion detection that relies on the database to detect intrusion. Another example is a BotNet DDoS intrusion detection with signature-based detection[2]. These IDSes can scan both IPv4 and IPv6-based intrusion easily[3].

However, this approach cannot recognize newer intrusions without regular updates[4]. The problem is exists in this kind of IDS. If the problem is not appropriately mitigated, many critical risks could occur. The first critical risk is an increase in server downtime. Since many businesses rely on servers to serve their customers, servers with more extended downtime may decrease customer numbers or move to better business[5, 6]. Rivaled businesses often use this action to gain customers while taking down others. The second critical risk is data loss. If an intrusion occurs during the transaction process, it could cause interruption[7]. Thus, the transaction process could be lost midway. Because of these risks, the business must treat the intrusion attacks seriously to prevent customer or data loss.

Numerous studies have used different approaches to detect intrusion and mitigate these problems, which are listed in Table 1. There were some studies that implemented supervised detection to identify intrusions with high accuracy compared to traditional models. In the following year, the detection model improved by combining two algorithms, such as a decision tree and gradient boosting, and was trained on a public dataset. Meanwhile, the most recent study designed a neural network model that was trained on a normalized and feature-selected dataset to achieve better accuracy. However, there were many problems with the previous studies like IPv4 training environment that is not compatible with IPv6 technology. Thus, the previous designed models cannot handle IPv6-based intrusions. The other problem is the behavior of the supervised learning algorithm, that lack of self improvement during the inference stage. Unless the supervised-based model is retrained with new data, it would not be able to detect new intrusions.

To mitigate the problem in the previous models, this study aims to design an intrusion detection system that can detect IPv6-based intrusions using a different approach. Instead of using supervised learning to detect intrusions, this study will utilize reinforcement learning (RL) with its online learning mechanism. Online learning in the RL algorithm does not need training first before detection and is capable of recording new intrusions immediately[8, 9]. Based on the article's review, they found that Reinforcement Learning (RL) is capable of optimizing policies for sequential decision-making problems, while Deep Reinforcement Learning (DRL) enables tackling more complex scenarios with large or high-dimensional data. The article also mentioned that both RL's effective policy optimization and DRL's scalability have clear advantages to enhance IDS[10].

An example is an article[11] in 2023 that successfully implemented Q Learning to detect IPv6 intrusions. However, the designed model has limited number of actions and cannot classify the intrusion based on the patterns. This study plans to improve the existing model in the article[11] by implementing State-Action-Reward-State-Action (SARSA) Learning to classify intrusion types based on their patterns. This study's novelty is using SARSA from reinforcement learning to classify IPv6 intrusions based on intrusion patterns. Besides that, this study is more focused on the behavior of the SARSA algorithm during classifying IPv6 intrusions, which was not reflected in previous studies.

2. Related Studies

In this section, the study creates a table containing previous related works that explain the current state-of-the-art of the study. This study explains the current State-of-The-Art in several subsections based on the detection method used in each study. There were traditional signature-based, supervised-based, and reinforcement learning. This study also discusses the weaknesses of previous studies that lead into this study. Table 1 contains the previous related works and their comparisons.

2.1 Traditional Signature-based Detection

This subsection explained the article with a traditional approach, like signature-based detection, which is usually implemented in Intrusion Detection Systems like Snort and Suricata. Article[12] implemented IDS like Snort, ModSecurity, and Nemesida and found that the detection rate on average up-to 83.26%, 73.77% and 63.89% unless ensemble approach is used. It was limited to three specific IDS and focuses exclusively on HTTP URI web attacks, which may limit the generalizability of the results to other attack types.

2.2 Supervised-based Detection

The next subsection explains all studies that implemented both classical and neural-based supervissed learnings. Article Kilincer was the earlier State-of-The-Art in this study, published in 2022, and tested many classical Machine Learning algorithms to detect Intrusions. Based on the evaluation, dataset CSE-CICIDS2018 has the highest accuracy up-to 99.92%. On the other side, Article Douiba designed a detection system combining gradient boost and a decision tree with an accuracy reaching 99.9%. The last classical machine learning detection was the article by Umar, where they proposed preprocessing to enhance the decision tree algorithm during detection. Their proposed model has an accuracy reached 99.75% for NSL-KDD and 98.51% for UNSW-NB15 datasets.

Besides classical machine learning, this study also found neural-network-based machine learning, as in Article [13] and [14]. The proposed model by [13] is designed to use a Lightweight Neural Network for the Internet of Things. Based on the evaluation, the model has an accuracy up-to 99.99% to detect Botnet and 98.94% to detect intrusion in the

Table 1. Previous Studies Comparison

Num	Literature	Key Feature	Technique	Result
1	Kilincer, I. F. et al. (2021)	Comparative analysis between supervised and unsupervised learning	Decision Tree, Random Forestk K-Means, K-Nearest Neighbors, Support Vector Machine, and XGBoost.	High Accuracy on CSE-CIC-IDS2018 dataset up-to 99.92%
2	Diaz-Verdejo, J. et al. (2022)	Web attacks using public web server and seven different attack datasets	Snort, ModSecurity, and Nemesida	Tshe average detection rates maxed out at just 83.26% for InspectorLog, 73.77% for ModSecurity, and 63.89% for Nemesida.
3	Zhao, R. et al. (2022)	Utilizing data dimensionality reduction for intrusion detection	Lightweight Neural Network	The model achieved an exceptional 99.99% accuracy on the Bot-IoT dataset and 98.94% on UNSW-NB15
4	Douiba, M. et al. (2023)	IoT security by utilizing an anomaly-based approach	Ensemble anomaly detection architecture	The GPU-accelerated model demonstrates exceptional efficacy, achieving Accuracy, Precision, and Recall metrics of approximately 99.9%
5	Daru, A. F. et al. (2023)	IPv6 flood attack detection based on epsilon greedy optimized Q learning in single board computer	Q-Learning with Epsilon Greedy Policy	The Q-Learning agent identified the intrusions with an accuracy up-to 98%
6	Schrötter, M. et al. (2024)	Investigates IoT network environments	Deep Neural Network and traditional Signature-Based	The replicated neural network model's accuracy plummeted to 54% when tested on new data from the exact same environment.
7	Umar, M. A. et al. (2025)	In-depth analysis of data preprocessing impacts the classification	Random Forest (RF) and Artificial Neural Networks (ANN/DNN).	The Random Forest algorithm demonstrated superior robustness, achieving peak accuracies of 99.75% and 98.51% on NSL-KDD and UNSW-NB15

UNSW-NB15 dataset. Meanwhile article by [14] designed a neural-network detection model and has high accuracy during detection.

The designed detection model in this subsection suffered several weaknesses like limited scope, a small dataset, and a classifier[15]. Gradient boosting in article [16] is often computation-intensive and prone to overfitting. The designed model by article[17] is still prone to a high level of false alarm rate despite using a modern dataset. Meanwhile, neural-network-based detection models also have some weaknesses. Light neural network models in article[13] usually performed well on simple attacks but may fail on more complex intrusions like IPv6. On the other side, the neural network model in article[14] suffered an accuracy drop to 54% on the new dataset with exactly the same environment.

2.3 Reinforcement Learning-based Detection

This subsection explained Reinforcement Learning-based detection. The article [11] has been successfully implemented, optimistic Reinforcement Learning using Q-Learning and Epsilon Greedy Policy. Based on the evaluation result, the designed Q-Learning agent capable to detect intrusion with an accuracy up-to 98%. However, the agent is only capable to differ between normal and intrusion and is unable to classify the intrusion type.

2.4 Research Gap and Relation to Proposed Solutions

However, the aforementioned models from previous years had several problems. The first problem concerns the protocol used to detect the intrusion. Most models only work in an IPv4 environment since either the signature dataset or pre-trained models are limited to IPv4 data. Thus, the previous models cannot handle IPv6 intrusions, increasing the danger of undetected IPv6-based intrusion. Furthermore, the past studies trained several models with existing datasets, and these data did not reflect the most recent intrusions. This problem fell into the population gap where the used datasets and their types

were not recent and lack of sufficient patterns. Thus, the past models cannot detect the most recent intrusions, especially in IPv6. This problem became critical since all ISPs must soon migrate their networking system to IPv6 addressing.

The second problem is the particular behavior in supervised learning. Previous studies must train supervised-based models before they can detect intrusions, a mechanism known as offline learning[18, 19]. Hence, this model will face a problem when detecting unknown or new intrusions. The model must be trained over and over to detect those new intrusions. This second problem fell into a methodological gap. Implementing supervised learning to detect intrusions is already standard in intrusion detection studies. However, the lack of self-improvement in supervised learning becomes a problem when facing new intrusions. Supervised-based model cannot recognize new intrusion unless re-training is involved. The training process in the supervised-based model was lengthy and wasted computing resources. The third problem that exists in the article Daru was about the implementation of the Q-Learning algorithm. The Q-Learning algorithm is an optimistic Reinforcement Learning and this algorithm tries to gain the highest reward possible. This behavior would lead to a higher false rate compared to a pessimistic RL algorithm like SARSA. In an Intrusion Detection system, a high false alarm rate is the most avoided aspect of a security system.

To address these problems in previous studies, this study proposes a detection model that diverges from neither traditional signature nor supervised-based methods. Unlike article[2] and article[20], this study implements a pessimistic Reinforcement Learning-based detection that allows the agent to detect intrusion during the inference stage. Meanwhile, the proposed detection agent is intended for an IPv6 environment, which previous models were unable to handle due to specification differences. Furthermore, this study also provides Reinforcement Learning behavior insights by testing the agent with different configurations, and validating with Shannon Entropy to calculate performance robustness on different seed configurations. To understand the behavior between optimistic and pessimistic Reinforcement Learning algorithms, this study compared the proposed agent with the Q-Learning agent from the article[11] and calculated the behavior on each episode.1

3. Methods

In this section, the study explains the step-by-step process from gathering data to designing a SARSA-based reinforcement learning agent. The explanation is separated into several subsections, such as study design, which explains what research methodology this study used to achieve its purpose. The subsequent subsection requires components and tools that explain what is required to gather the data, followed by the data gathering subsection, which explains the data gathering process. The final subsection is Analysis, which explains how this system evaluates and validates the results from the intrusion detection agent.

3.1 Study Design

This subsection explains the study design used to achieve a SARSA-based IPv6 intrusion detection agent. This study used an experiment-based design by creating an isolated situation with controlled variables to understand how IDS and RL-based agents work. The first step is to identify the problem and review existing literature. This study found a problem with the detection approaches in previous IDS models based on the literature. Since the previous models depend on signature-based detection (that requires regular updates) or supervised-based detection (that requires retraining), those models cannot improve themselves because of algorithm limitations[21]. Hence, this study aimed to design a self-improving intrusion detection system with an RL-based algorithm (in this case, State-Action-Reward-State-Action or SARSA).

After identifying the problem, this study proceeds to the next step: creating the experiment. An isolated network topology with IPv6 configuration is the core of the experiment. This configuration allowed this study to generate IPv6-based intrusions without any mix-up with random data. Besides that, an isolated configuration allowed the RL agent to focus more on intrusion patterns. After the experiment stage, this study continues the process of creating the RL agent. The agents have several customizations, and the whole explanation is available in the subsequent subsection. After the agents are completed, this study evaluates and validates the results to ensure their robustness.

3.2 Required Components and Tools

This subsection explains the required components and tools to ensure the reproducibility of the agents in future studies. To gather the required data, this study used several publicly available software, such as The Hacker Choice IPv6 (or THC-IPv6)[22] to generate IPv6-based intrusions and Wireshark[23], Gymansium as the RL agent's framework[24], and Python IDE to write RL agents. This study used two virtual machines installed with the Ubuntu operating system to create an isolated environment and connect locally. With this experiment configuration, this study is ready to gather intrusion data before designing an RL agent with the SARSA algorithm. Figure 1 illustrates the topology used to gather the required data. The left side is the attacker with the THC-IPv6 tool installed. Meanwhile, on the right side is the target with Wireshark to gather all intrusions from the attacker[18, 19].

3.3 Data Gathering

This subsection explains the data-gathering process, the intrusion type, and the sample numbers used for online training with the designed RL agents. While THC-IPv6 did provide a significant number of IPv6-based intrusions, not all tools are required in this case. The used tools in this study were denial6 (Opt 1, 2, 4, and 5), flood_mld26, flood_redir6, flood_rs6, flood_solicitate6, flood_unreach6, ndpexhaust26, rsmurf6, and seendpees6. These tools generated ICMPv6 traffic with high-speed attack speed to the target. With the flood, speech reaches 1000 packets per second to the target, sufficient to slow down

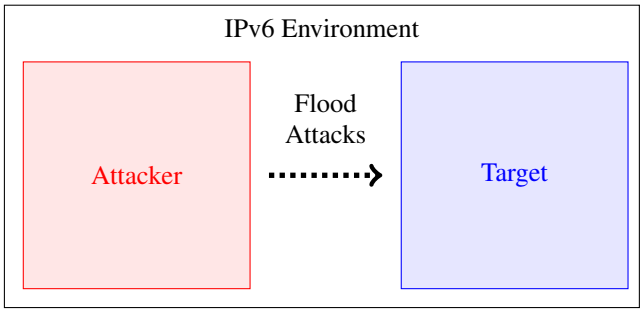

Figure 1. Data gathering topology illustration

the target's network connectivity. After that, this study gathered five million sample rows to ensure that every pattern within the intrusions was captured with Wireshark. However, this study cannot use these data as they are; they need further preprocessing and cleaning. Before the preprocessing and cleaning processes start.

3.4 Preprocessing

After gathering the intrusions, the process continues to the next step: the preprocessing and cleaning stage. This stage is critical for reinforcement learning agents before the online learning phase starts[25, 26, 27]. Similar to supervised-based learning that only works with numeric data, RL agents require numeric type data for the learning process. This data type can be obtained by preprocessing and cleaning raw capture files from Wireshark into a proper table.

There were several steps in the preprocessing phase such as: convert *null* or (NaN) data into unique number (in this case is 99999) and convert hexadecimal to decimal value. The last preprocessing step extracts the necessary fields to create unique patterns that distinguish one intrusion from another. This study uses 11 specific columns to capture each intrusion's uniqueness: frame length, Ethernet type, IPv6 payload length, IPv6 next header, ICMPv6 type, ICMPv6 code, ICMPv6 checksum, ICMPv6 echo sequence number, data length, and data.

After a preprocessing stage, the dataset is ready for the subsequent process: cleaning. To extract the patterns available in each intrusion, this study removes duplicates within the dataset. The cleaning phase was recommended and often used in cyber forensic[28]. This approach can be done easily with any spreadsheet-based software or through Python scripting. After duplicate removal, this study obtained a unique pattern of each intrusion shown in Table 2

According to Table 2, these patterns are the deduplication results from five million intrusion samples. Many intrusions, such as Denial6, Sendpees6, and Rsmurf6, have only one pattern. On the other hand, some intrusions, like NDPExhaust26 and Flood_mld26, have variations in their patterns. The total number of all patterns was 218,467 patterns. Since this study took a significant number of samples, all patterns available in each intrusion were successfully recorded. At this point,

Table 2. Intrusion Number of Patterns

Intrusion Name	Options	Number of Pattern
Denial 6	Case 1	1
Denial 6	Case 2	1
Denial 6	Case 4	1
Denial 6	Case 5	1
Flood_mld26	-	65,536
Flood_redir6	-	21,846
Flood_rs6	-	1
Flood_solicitate6	-	65,536
Flood_unreach6	-	1
NDPExhaust26	-	65,541
Rsmurf6	-	1
Sendpees6	-	1
Total Patterns		218,467

the dataset for the agent's online training was ready and this study continued to design the reinforcement learning agent using the SARSA algorithm.

3.5 Reinforcement Learning Agent

This subsection explains the designing process of the RL agent. The process in this stage involves writing a Python script that imports the Gymnasium library[24] as the framework for the RL agent. This algorithm often used in robotics such as unmanned aerial vehicle (UAV)[29, 30]. In this case, this study designed State-Action-Reward-State-Action (or SARSA in short) Learning agent for the main RL agent. However, during the design process of the RL agent, there were several changes occurred to adapt RL agent behavior to classify IPv6 intrusions. Usually, the RL environment refers to the physical environment which is converted to a virtual environment[31].

Since IPv6 intrusions only appear in the digital world and not the real world, then the environment for this situation also changed as well. The agent is configured to read IPv6 patterns and then create decisions based on that pattern. If the agent correctly chooses an action, the environment will reward the agent with five (5) points. On the contrary, then the agent will receive negative five (-5) points if the action is incorrect. Balanced reward was required in order to avoid bias[32, 33]. Epsilon Greedy is a probability method for RL agents to decide an action based on percentage. With this method, an RL agent may choose between exploration (random choice) or exploitation (use memorized information)[34, 35]. However, this study used three different Epsilon Greedy configurations for different RL agents' behavior. The first configuration used 0.1 where this configuration means that the RL agent will prefer exploitation more than exploration. The second configuration used 0.5 where the action decision is balanced between exploitation and exploration. The last configuration used 0.9 where the action decision lean on exploration more than exploitation.

Figure 2 explains how the agent interacts with intrusion data and the environment. The classification process starts with the agent iterating the dataset one by one, the dataset ex-

Algorithm 1 SARSA Algorithm Pseudocode

Input: Dataset D, Learning Rate α, discount factor γ, exploration rate ε
Output: Trained Q-function $Q(s,a)$
Initialize $Q(S,a)$ for all $s \in S$, $a \in A$
for do*episode* = 1 to 50, do
    Initialize state s from dataset D
    Choose action a from s using ε-greedy policy derived from q
    for e **do**ach step in *episode*, do
        Execute action a, observe reward r and next state s'
        Choose action a' from s' using ε-greedy policy from Q
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \cdot Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$; $a \leftarrow a'$
        if s is terminal **then**
            break
        end if
    end for
end for
return Q

tracted its row to the agent. With that data row, the agent read the pattern as the environment state and created the decision based on the Epsilon Greedy configuration. After that, the agent will send the decision to the environment and receive the reward from the environment. The RL agent calculated the reward and stored the result in the Q Table. The process continues until the dataset is fully iterated for one epoch. In this study, an epoch refers to one episode of agent training session consisting of a fixed number of interaction loops (in this study, limited to 50 episodes). In each interaction loop, the Reinforcement Learning agent iteratively observes the state (intrusion data from dataset rows), selects an action based on its policy (exploration or exploitation, depending on epsilon configuration), receives a reward, transitions to the next state, and updates the Q-table according to the SARSA update rules. In the next epoch, learning progress is evaluated to monitor convergence and stability. This study decided to limit the maximum epoch to 50 loops. This number is deemed sufficient after the author monitors the convergence and the stability of total rewards obtained by the agents. To ensure reproducibility, this study adds pseudocode in Algorithm 1.

Algorithm 1 demonstrates a basic SARSA algorithm. The process begins by loading a custom environment and dataset, followed by initializing an empty Q-table. After the initialization process, the agent then starts the first loop of the epoch. In the first loop, the agent started iterating through the dataset rows as the state. Once the agent gets the state, the next step is to decide the action based on an epsilon-greedy policy. The epsilon controls whether the action is from exploration (random action from environment space) or exploits the existing Q Table state-action value. The agent then passes the state and

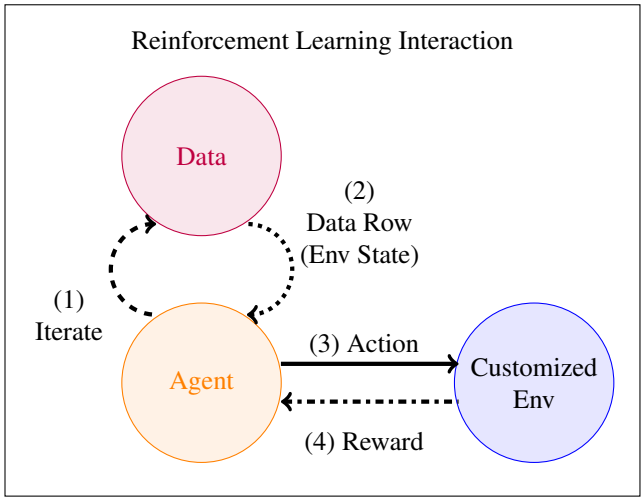

Figure 2. Agent's interaction with environment

the action to the environment, and the environment returns the reward. If positive, this means the action is correct and vice versa. The agent then continues to calculate the Q value by multiplying the learning rate and the temporal difference of reward, discount factor, next state Q value, and current state Q value.

Despite using a static dataset instead of a live dataset, the agent processed the sample row-by-row sequentially in the order in which they were captured. This approach is intended to simulate live interaction where each network packet represents a single timestep of the environment. Thus, this approach is different from pure offline methods, where the supervised model is capable of feeding numerous data rows at once. An RL agent is only capable of processing one data row as one state and deciding an appropriate action based on that state.

3.6 Analysis

The last subsection in the Method section is Analysis. In this subsection, this study explains the evaluation and validation processes. To evaluate the RL agent, this study used several approaches. The first evaluation compared different Epsilon Greedy configurations. In this case, there were three Epsilon Greedy configurations: 0.1 (named with E-0.1), 0.5 (E-0.5), and 0.9 (E-0.9). This study compared the total number of correct and incorrect actions produced by each agent to display which agent had the most correct and the least incorrect actions. Besides that, this study also compared the total and average rewards per epoch.

The total reward will show whether the agent has high accuracy while detecting intrusions. In this case, the result should be positive. However, if the result was negative, then the agent performed poorly. The average reward per epoch reflected the average obtained reward for every action the agent decided per epoch. Similar to the total reward, if the result was positive, the agent performed excellently and vice versa.

The second evaluation is a comparison with Q Learning algorithms based on the previous model in the article[11]. Both algorithms were RL algorithms but different in reward policies and behavior. The algorithm used by the previous agent utilized off-policy. This agent started learning the optimal value independently based on the current policy. On the contrary, on-policy allowed the agent to learn the optimal value based on what its do. To determine the behavior of each algorithm, this study used the following equation to obtain the mean of state values shown in Equation 1.

$$\bar{V} = \frac{1}{N}\sum_{i=1}^{N} V(s_i) \tag{1}$$

Where Equation 1 consists of $\bar{V}$ as the mean of state value, N is the number of states, and $V(s_i)$ is the value in the i state. This equation will determine if the algorithm is leaning towards optimistic or pessimistic. If the means are higher, then the algorithm leans towards optimism. On the contrary, a lower mean indicated the algorithm leaned into a realistic (pessimistic)

This study also used a statistical approach with Shannon Entropy to validate the results, calculating the action policy for every step taken and performance robustness via different random seeds. The Shannon Entropy calculation displayed the action's convergence and often used in machine learning's implementations[36, 37, 38]. If the entropy result was closing near 0, then the agent successfully converged its learning process. Conversely, if the agent fails to reach near zero, the agent is still exploring the available action. Equation 2 reflected the calculation of Shannon Entropy.

$$E(prob(\cdot|s)) = -\sum_{a\in A} prob(a|s) \times \log_2 prob(a|s) \tag{2}$$

In Equation 2, the equation of Shannon Entropy consists of a, which reflects the action; s, which refers to the state; and E, which is the result of entropy. The other evaluation is to calculate the total reward of agent E-0.1, but random seeds handled the variation[39, 40, 41]. This study used five random seeds such as 137318, 191446, 186765, 693667, and 831766. These numbers were pseudo-random, and this study utilized a random function from Python's library to generate these numbers. The results produced by these seeds are then compared side-by-side to display the robustness of agent E-0.1.

4. Results and Discussion

This section explains the study results based on RL agents evaluations. This study presented several figures that illustrated the RL agent's performance per Epoch times. The performance evaluations consist of performance comparison between Epsilon Greedy configurations in terms of total correct and incorrect actions, and total rewards. Besides that, there was a performance comparison with the Q-Learning model.

4.1 Results

The result from the proposed agent evaluation is the comparison with three different Epsilon Greedy configurations. The first comparison is the total correct action between SARSA agents. Figure 3 illustrates the total correct actions between agents E-0.1, E-0.5, and E-0.9.

Figure 3 illustrates the evaluation result for the total correct actions for each SARSA agent with different Epsilon configurations. According to the result, the SARSA agent with Epsilon configured to 0.1 (E-0.1 in short) has the fastest epoch to reach the highest total correct action numbers. In the eleventh step, the E-0.1 agent reached a stable value within 193,453 correct actions with minor variations. The followed by E-0.5 at the 18th step with a total correct action of 117,720. In last place was E-0.9 at the 23rd step with a total correct action of 37,254.

The second result after total correct action was total incorrect actions. This result contains accumulative incorrect actions made by the agent per epoch. Figure 4 contains the result of total incorrect actions per epoch. According to Figure 4 the results in this category are opposite to those in Figure 3. Thus, agent E-0.1 has the lowest number of incorrect actions, as many as 25,014 at the 11th step, followed by E-0.5 with 100,747 at the 18th step and E-0.9 with 181,213 at the 23rd step. These results (both correct and incorrect actions) would affect the total reward obtained in the subsequent evaluation.

The third result is the total reward obtained by each agent per epoch. The total reward is obtainable by multiplying correct actions by five (5) points for total correct rewards. Then, calculate the total incorrect rewards by multiplying incorrect actions by negative five (-5) points. The sum of the total correct and incorrect rewards equals the total agent reward for classifying IPv6 intrusions. Figure 5 illustrates the total reward for each agent.

According to Figure 5, all agents start with negative rewards at epoch zero. This situation was typical for the agents in the early online training phase. After progressing through the learning process per epoch, this number increases positively. Based on the result, agent E-0.1 successfully reached a positive total reward of 842,195 at the 11th epoch. Agent E-0.5 is in second place with a total reward of 84,865 at the 18th epoch. Although agent E-0.5 has a positive number, the result is lower than E-0.1. The last place is agent E-0.9, with a total reward of -719,795 at the 23rd epoch step. Agent-0.9 was the only agent with a negative total reward and performed poorly. According to the result, agent E-0.1 performs the highest among the other agents.

The following result is the agents' average reward per epoch. Unlike total reward, which is calculated as the overall reward, this study obtained the average reward by summing all rewards within an epoch and dividing them by total intrusion numbers. Similar to total rewards, if the average reward per

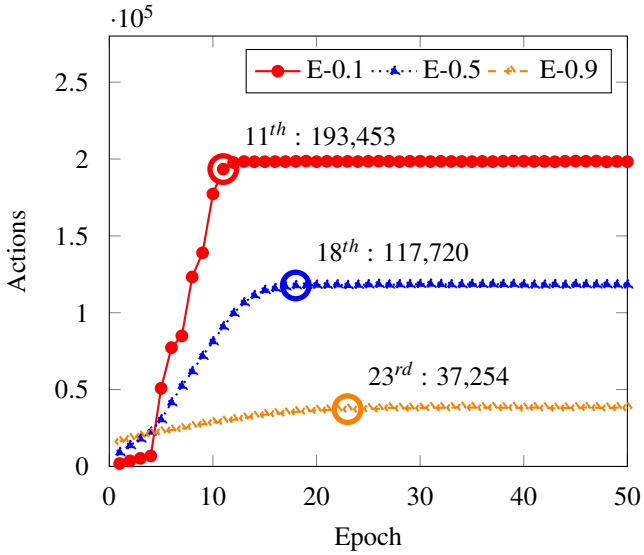

Figure 3. Total Correct Actions comparison between Epsilons

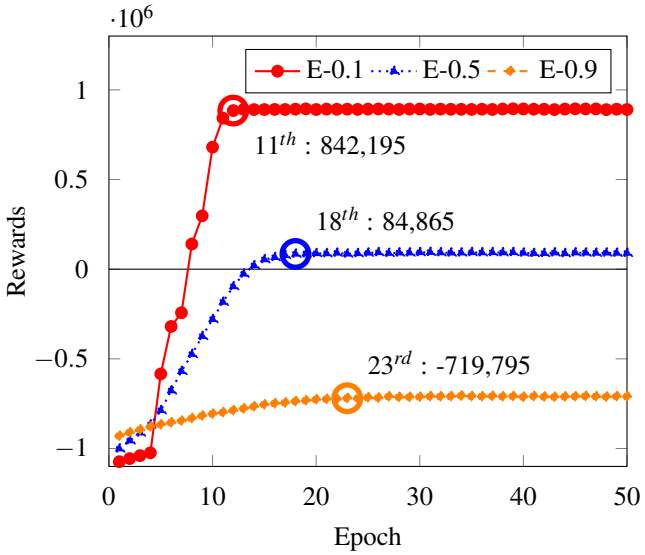

Figure 5. Total Reward comparison between Epsilons

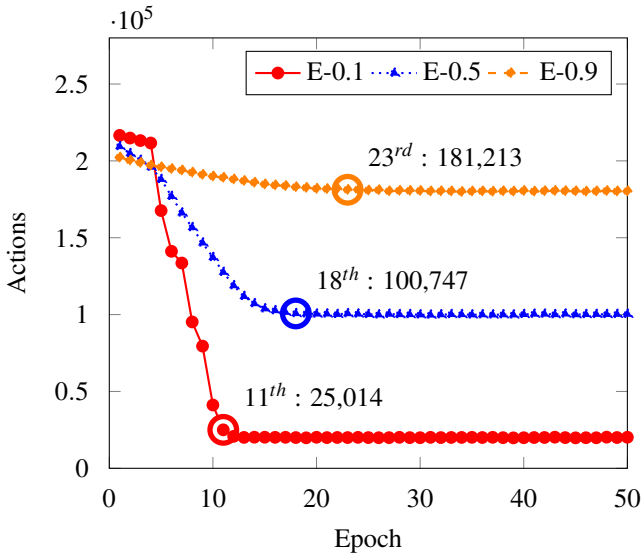

Figure 4. Total Incorrect Actions comparison between Epsilons

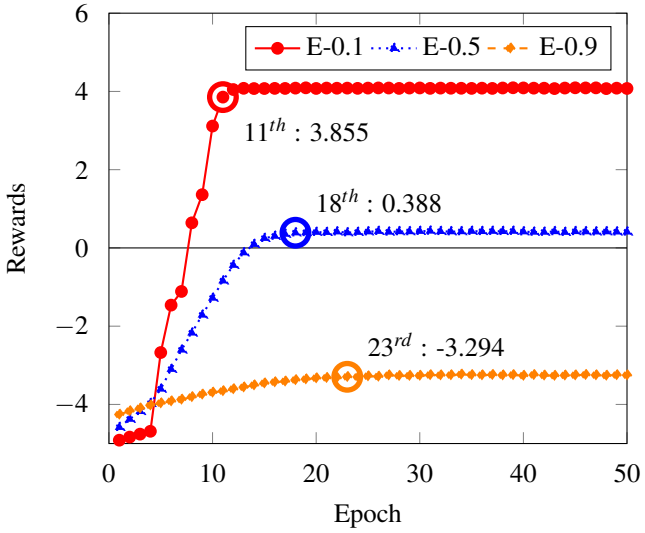

Figure 6. Average Reward per Epoch

epoch was positive, then the agent growth was excellent, and vice versa. Figure 6 illustrates the average reward per epoch for all agents.

According to Figure 6, the average reward for agent E-0.1 was 3.855 per epoch starting at the 11th epoch. Agent E-0.5 had a 0.388 average reward per epoch starting at the 18th step. In last place, E-0.6 had a -3.294 average reward per epoch at the 23rd step. Based on Figure 6, this result proved SARSA Agent E-0.1's performance compared to other SARSA agents with different Epsilon configurations.

After behavioral analysis, the performance of the SARSA agent was evaluated. Measurements using the psutil library showed that the agent consumed 25% of the computer's total CPU resources. In terms of memory usage, SARSA required 1.08 GB of memory, while virtual memory usage reached 2.49 GB. For inference, the SARSA agent required 1.9 ms to obtain the reward from the environment and compute the Q value.

The following result compares with the agent based on the article[11]. This study configured both agents to use the same configuration and behaviour except for the RL algorithms. The total rewards from both agents were then compared side to side to display which was better. Figure 7 illustrates the total rewards comparison between the Q-Learning agent from article[11] and the proposed SARSA agent.

According to the comparison result in Figure 7, this study found that both agents' results were similar. Especially when this study tested both algorithms by purely exploiting the Q

Table (epsilon set to zero). Both algorithms produced exactly the same total correct and incorrect classifications. For that reason, this study analyzes in depth by comparing each state in the Q Table from each algorithm and found several facts. Based on the mean difference from both algorithms, there was a -1.2109 average difference, with the SARSA algorithm lower than Q-Learning. This result showed that the Q-Learning algorithm is more optimistic than the SARSA algorithm. To add more proof to this result, this study calculated the mean of state values (Vs) from each Q Table and found that Q-Learning has a higher mean V(s) up to 16.310. On the contrary, SARSA has a mean V(s) up to 4.903. A high mean V(s) indicated that Q-Learning is highly optimistic and willing to risk more for a higher reward. Meanwhile, SARSA tried to be more realistic (pessimistic).

After explaining the evaluation results, this study continues to explain the validation results. Two validations were done, such as Action Entropy validation to understand each agent's action policy and Random Seed validation to test the agents' robustness against random seed.

The first validation result was the action entropy from Shannon Entropy. Figure 8 illustrates the action entropy result between SARSA agents with different Epsilon configurations. Based on the validation in Figure 8, SARSA agent E-0.1 showed the fastest stability at the 25th step and the lowest entropy of 0.0904 compared to other agents. This result showed that the proposed SARSA agent reached action stability faster than other agents. SARSA agent E-0.5 is the second fastest after E-0.1, with an entropy score of 0.1886 at the 44th step. The last place was agent E-0.9, which cannot reach a stable action policy after the 50th epoch.

The last validation result compares a similar agent with a different random seed. This validation verifies the changes in RL behaviour and whether they are affected by random seeds. Figure 9 illustrates the performance comparison between SARSA agents with different random seeds. According to Figure 9, the results from all random seed agents are almost similar, with a minimal margin of difference.

4.2 Discussions

This subsection discusses the entire evaluations, validations, implications, and future studies of this topic. This discussion is an important part of explaining every result in detail. The first discussion is about the result interpretation obtained by this study. This study evaluated the agents' performance in several aspects, such as total correct and incorrect actions, total and average reward, and comparison with other RL algorithms.

Based on these evaluations, the proposed SARSA agent with Epsilon E-0.1 configuration has the highest performance compared to other SARSA. This agent had 193,453 correct actions at the earliest epoch step, compared to E-0.5, which had 117,720 correct actions at the 18th step, and E-0.9, which had 37,254 at the 23rd step. Since the result of total correct action is clear, the result of incorrect action is also reflected. Agent

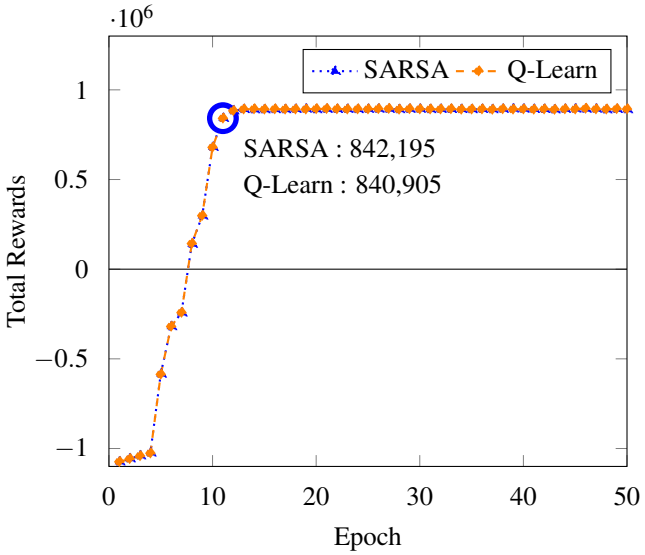

Figure 7. Total Reward comparison between Algorithm

E-0.1 had the lowest incorrect action, followed by agents E-0.5 and E-0.9. The total reward result is the summation of the correct action multiplied by the positive reward and the incorrect action multiplied by the negative reward. This result showed whether the agent performed excellently or not. Based on the result, agent E-0.1 performed excellently with the highest total reward. Followed by agent E-0.5 with a modest but positive total reward.

However, agent E-0.9 performed poorly, with negative total rewards in this case. The average reward reflected the average of the reward received by the agent from all patterns. These data are collected per epoch to show whether the agent received positive or negative rewards per epoch. According to the result, agent E-0.1 received the stable reward of 3.855 per epoch at the 11th step. This was followed by agent E-0.1 with an average reward of 0.388 and E-0.9 -3.294 per epoch. What made reinforcement learning unique was the pattern of the graphs in Figures 3, 5, and 6. Although each figure refers to a different evaluation, the pattern is similar. This phenomenon occurred because total correct action, total reward, and average reward per epoch were related. Performance evaluation indicates that the SARSA agent required 25% of total CPU resources, 1.08 GB of memory, and 2.49 GB of virtual memory. These resource requirements are considered acceptable trade-off in the context of modern computing systems.

The second discussion concerns the result comparison with the previous model or agent, the Q-Learning agent from the article[11]. When compared to the proposed SARSA agent and the Q-Learning agent, both algorithms performed similarly, especially when the policy is set to exploit the Q-Table. However, after in-depth analysis by calculating the state values in each Q-Table, this study found several facts on how each algorithm behaves. Based on the result, there was a -1.2109 mean difference between SARSA and Q-Learning. SARSA has a lower difference compared to Q-Learning. This

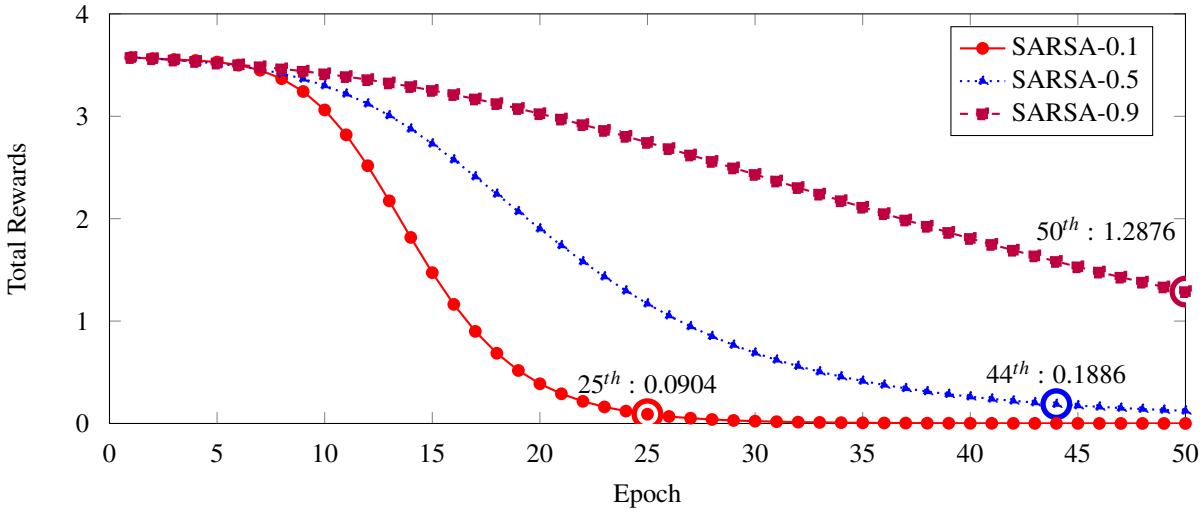

Figure 8. Action Entropies between Epsilon

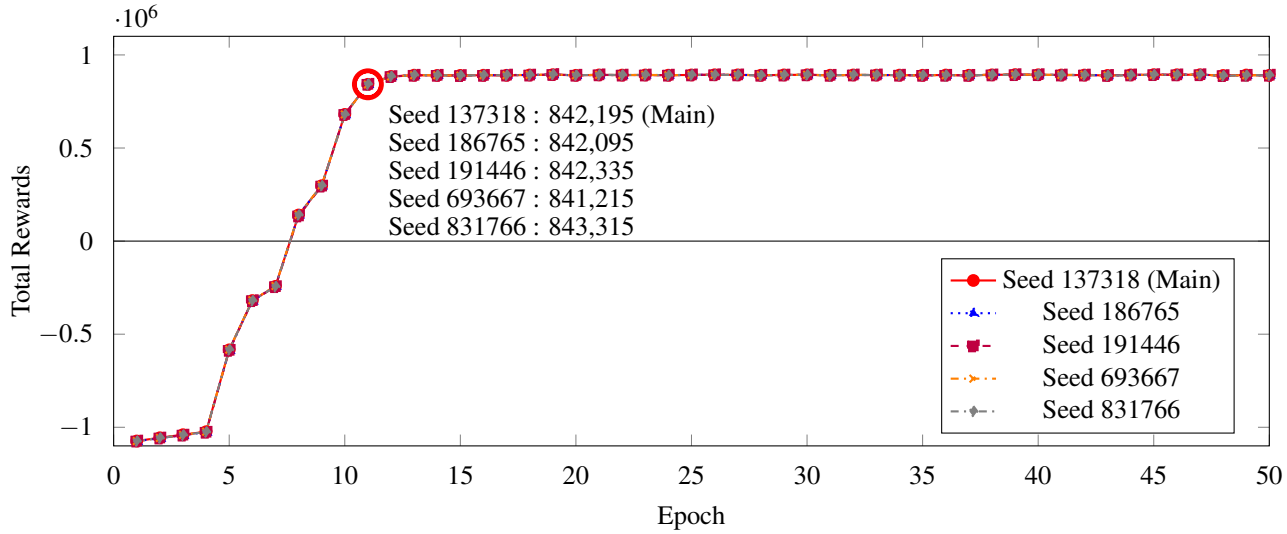

Figure 9. Performance Validation with Random Seeds

result meant that SARSA used a realistic (pessimistic) approach when trying to obtain the reward.

Compared to Q-Learning, which is more aggressive in getting a higher reward. By calculating mean V(s) for each algorithm, this study measured how optimistic an RL algorithm was. Q-Learning was the most optimistic, with a mean V(s) of 16.310, and SARSA was the least optimistic (pessimistic), with a mean V(s) of 4.903. When an algorithm behaves more optimistically, this behavior will lead to faster learning but higher false alarms by flagging normal packets as intrusions. There was also a problem if implemented in a stochastic network environment, where optimism amplifies error. On the contrary, SARSA has the benefit of stable and reliable policy, lower false positive rate, safer real-world deployment, but suffers from slower learning. Despite the advantages and disadvantages of each algorithm, SARSA is the most preferred in a security context due to lower false positive alarms.

This study did not compare the proposed SARSA agent result with supervised-based learning because the algorithm's approach differed fundamentally. The supervised-based model requires training before testing or deployment, while RL-based agents can be used almost immediately without training. In the supervised-based model case, the model tries to predict, based on mathematically, a learned pattern. On the contrary, the RL-based agent recorded the pattern as it is without any mathematical process. Thus, it is easier to reach high accuracy for RL-based agents. Because of that, this study will not compare their performance.

The third discussion is the implication for both theoretical and practical sides. In theoretical implication's side, it is typical for an Intrusion Detection System to use either supervised-based learning or signature-based detection. However, the

approach to using Reinforcement Learning itself is rare. Thus, this study created an insight to utilize something in its default configuration from Reinforcement Learning. Conversely, the practical implication of Reinforcement Learning refers to its usage. Reinforcement Learning is often used in robotics to do regular or dangerous tasks related to physical activities. Thus, implementing an RL agent as an Intrusion Detection System and doing cyber activities is innovative.

The fourth discussion concerns the proposed agent's strengths. The agent's strength is several points. The first point is the speed of learning. Unlike a supervised-based model that requires longer training time, an RL-based agent can learn within a shorter time. This study only needs 50 epochs to create a high-performance agent, while supervised-based learning might need more than 100 epochs. The second strength point is the capability to learn new patterns almost immediately. RL-based agents like Q-Learning or SARSA memorize the pattern and store it in a table. Thus, the agent knows that the pattern exists. Meanwhile, a supervised-based model must be retrained to recognize new patterns. The third point is further optimization; the RL algorithm can be combined with supervised learning like neural or deep neural networks. This combination will create Deep Reinforcement Learning with both supervised and reinforcement learning features.

Based on these discussions, this study concluded that the proposed SARSA-based intrusion detection agent performs better than the previous agent. The action policy validation through Shannon Entropy showed fast convergence with Epsilon 0.1 configurations. The SARSA agent also has consistent performance with a small margin and is unaffected by random seed variations.

5. Limitations and Future Study

This section explains the limitations and potential improvements of the current proposed solution. This study presents the limitations and future study in two subsections. The first subsection explains the limitations of the proposed solutions. The first limitation is the mechanism of the RL-based agent. RL-based agent requires an environment to verify whether its decision is correct. Thus, the agent cannot act independently as the supervised-based model does.

The second limitation is the complexity of building an agent. Supervised learning is easy to build compared to RL-based agents. Before creating an agent, this study must first design the environment and its mechanism. Then the agent comes after that. The third limitation of the proposed agent was a practical implementation regarding real-world deployment. All RL agents in this study were developed in Python using the standard gymnasium library. This language is an interpreted language that has higher computational overhead compared to the highly optimized C++ implementations used in production-level IDS like Snort or Suricata. This limitation also affects network-based metrics like throughput and packet loss, which were not evaluated in this study, as the current Python-based prototype is not yet suitable for high-speed network environments.

The fourth limitation of the proposed agent was the reliance on a controlled, isolated laboratory environment for data generation. While the use of the THC-IPv6 toolkit provided a necessary baseline for observing RL agent behavior, the resulting agent may not fully capture the stochastic nature and high-dimensional noise found in production-level IPv6 networks. The fifth limitation in the proposed RL agent is based on its use of a static, pre-defined feature vector for its state space representation. In IPv6 networks, traffic behavior is highly dynamic, frequently resulting in multi-dimensional feature drift. If the network environment dynamically shifts or introduces new dimensional features, the fixed discretization boundaries become obsolete, leading to a state-space explosion Q.

The second subsection concerns the future study of RL-based intrusion detection systems. The proposed SARSA agent can be optimized by adding supervised or unsupervised learning. Combining RL with unsupervised learning will produce an agent capable of improving and self-learning. To address concerns regarding generalizability, future studies will emphasize evaluating the RL agent across several datasets. This includes the combination of modern, mixed-protocol benchmarks. To handle dynamic multi-dimensional features, this study plans to first extract IPv6 traffic features, then apply a dimension-reduction technique like Principal Component Analysis (PCA). The resulting low-dimensional, dense state space will be used as input for the tabular SARSA agent, allowing it to maintain fast convergence times even as network complexity scales.

Improving the performance of the proposed SARSA agent can be achieved by implementing C++, which performs better than Python and can significantly improve performance. However, writing an intrusion detection system based on Reinforcement Learning from scratch is not an easy task and requires deeper knowledge of the language. On the other hand, immediately implementing the proposed solution into a real-life environment introduces significant operational problems. Operating system binary support, system call availability, and hardware support are the most critical problems that this study must face and cannot be instantly achieved. Iterative and rigorous developments in the future study are required to create production grade intrusion detection system.

6. Conclusion

The intrusion detection system is a vital part of a healthy network. Without this software, a network will receive an intrusion attack from an unknown third party, which can cause dangerous risks to the internal system. This situation can be avoided if an intrusion detection system is installed. There are many IDS variations, such as signature-based and supervised-based systems. According to the analysis, both variations have weaknesses regarding the dataset and self-improvement approach.

A signature-based model requires regular updates to rec-

ognize new intrusions, while the supervised-based model requires retraining. Because of that, this study aimed to design an intrusion detection system based on an existing model. Instead of using supervised learning to detect intrusion, this study used State-Action-Reward-State-Action or SARSA Learning to create an agent to detect IPv6-based intrusions. Based on the evaluation and validation results. This agent with the epsilon 0.1 configuration (E-0.1) has the fastest and highest performance compared to another agent with different epsilon configurations. Agent E-0.1 reached the highest at the 11th step with 193,453 correct actions, 25,014 incorrect actions, 842,195 total rewards, and 3.855 average reward per epoch. Agent E-0.1's result is higher than agent E-0.5 and E-0.9.

A comparison of the proposed RL agent using SARSA and the previous Q-Learning agent shows that both algorithms yield similar results when the Q-Table is fully exploited. However, detailed analysis of the Q-Table indicates distinct behavioral differences between the two algorithms. Calculations show a -1.2109 difference between SARSA and Q-Learning, with SARSA exhibiting a lower value. The mean state value V(s) also differs substantially, with SARSA at 4.903 and Q-Learning at 16.310. Higher mean state values suggest more optimistic behavior, whereas lower means reflect more realistic (pessimistic) tendencies. More optimistic behavior leads to a higher false alarm rate due to the agent trying to get the reward as high as possible. On the contrary, SARSA learning has a stable and lower false alarm rate. Due to the lower false alarm rate, SARSA is more preferable in a security context.

These evaluation results were then validated with random seed variation to see the robustness of the result. Based on the validation result with random seed variation, the agent of E-0.1 has a consistent total reward with a small margin. This result proved that the RL algorithm has stable decision-making compared to the supervised learning model. This study also calculated the action policy with Shannon Entropy to prove this result. According to the validation, the E-0.1 agent reached the convergence faster than the other agent. Thus, this study concluded that the proposed SARSA agent performed better than the previous Q-learning model with more patterns recognized.

Acknowledgements

The authors sincerely acknowledge the sponsorship and support extended through the international collaboration between the Kulliyyah of Information and Communication Technology, International Islamic University Malaysia, and the Faculty of Information and Communication Technology, Universitas Semarang.

Author contributions

This study has separated each author's role. Starting in April, Firman Daru did conceptualization, funding acquisition, and supervision. Alauddin Maulana Hirzan did the data curation, software, validation, and writing the original draft. Then followed by Zainab Senan Mahmod Attar Bashi who did investigation, resource, and review the draft. The last author, Fajriannoor Fanani, did the formal analysis, visualization, and draft editing.

Declaration of AI-Assisted Use in Writing

The authors declared that during the writing the manuscript, they utilized AI tools to assist in correcting grammatical errors within the manuscript.

References

[1] M. Tajdini; H. Kolivand. IPv6 Detection Techniques and Solutions. In: *2023 16th International Conference on Developments in eSystems Engineering (DeSE)*. [S.l.: s.n.], 2023. p. 663–666. Journal Abbreviation: 2023 16th International Conference on Developments in eSystems Engineering (DeSE).

[2] SZYNKIEWICZ, P. Signature-Based Detection of Botnet DDoS Attacks. In: KOŁODZIEJ, J.; REPETTO, M.; DUZHA, A. (Ed.). *Cybersecurity of Digital Service Chains: Challenges, Methodologies, and Tools*. Cham: Springer International Publishing, 2022. p. 120–135. ISBN 978-3-031-04036-8. Disponível em: ⟨https://doi.org/10.1007/978-3-031-04036-8_6⟩.

[3] LIU, N. et al. A Survey on IPv6 Security Threats and Defense Mechanisms. In: SUN, X. et al. (Ed.). *Artificial Intelligence and Security*. Cham: Springer International Publishing, 2022. p. 583–598. ISBN 978-3-031-06794-5.

[4] T. Alsmadi; N. Alqudah. A Survey on malware detection techniques. In: *2021 International Conference on Information Technology (ICIT)*. [S.l.: s.n.], 2021. p. 371–376. Journal Abbreviation: 2021 International Conference on Information Technology (ICIT).

[5] KESKIN, O. F. et al. Cyber Third-Party Risk Management: A Comparison of Non-Intrusive Risk Scoring Reports. *Electronics*, v. 10, n. 10, 2021. ISSN 2079-9292.

[6] CREMER, F. et al. Cyber risk and cybersecurity: a systematic review of data availability. *The Geneva papers on risk and insurance. Issues and practice*, v. 47, n. 3, p. 698–736, 2022. ISSN 1468-0440 1018-5895. Place: England.

[7] ASLAN, O. et al. A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions. *Electronics*, v. 12, n. 6, 2023. ISSN 2079-9292. Disponível em: ⟨https://www.mdpi.com/2079-9292/12/6/1333⟩.

[8] HOI, S. C. et al. Online learning: A comprehensive survey. *Neurocomputing*, v. 459, p. 249–289, out. 2021. ISSN 0925-2312. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0925231221006706⟩.

[9] GREENHOW, C. et al. Foundations of online learning: Challenges and opportunities. *Educational*

Psychologist, v. 57, n. 3, p. 131–147, jul. 2022. ISSN 0046-1520. Publisher: Routledge. Disponível em: ⟨https://doi.org/10.1080/00461520.2022.2090364⟩.

[10] H. Kheddar et al. Reinforcement-Learning-Based Intrusion Detection in Communication Networks: A Review. *IEEE Communications Surveys & Tutorials*, v. 27, n. 4, p. 2420–2469, ago. 2025. ISSN 1553-877X.

[11] DARU, A. F.; HARTOMO, K. D.; PURNOMO, H. D. IPv6 flood attack detection based on epsilon greedy optimized Q learning in single board computer. *International Journal of Electrical & Computer Engineering (2088-8708)*, v. 13, n. 5, p. 5782–5791, 2023. ISSN 2088-8708. Disponível em: ⟨https://ijece.iaescore.com/index.php/IJECE/article/view/30100⟩.

[12] DíAZ-VERDEJO, J. et al. On the Detection Capabilities of Signature-Based Intrusion Detection Systems in the Context of Web Attacks. *Applied Sciences*, v. 12, n. 2, 2022. ISSN 2076-3417.

[13] ZHAO, R. et al. A Novel Intrusion Detection Method Based on Lightweight Neural Network for Internet of Things. *IEEE Internet of Things Journal*, v. 9, n. 12, p. 9960–9972, jun. 2022. ISSN 2327-4662.

[14] SCHRöTTER, M.; NIEMANN, A.; SCHNOR, B. A Comparison of Neural-Network-Based Intrusion Detection against Signature-Based Detection in IoT Networks. *Information*, v. 15, n. 3, 2024. ISSN 2078-2489.

[15] KILINCER, I. F.; ERTAM, F.; SENGUR, A. Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Computer Networks*, v. 188, p. 107840, 2021. ISSN 1389-1286. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1389128621000141⟩.

[16] DOUIBA, M. et al. Anomaly detection model based on gradient boosting and decision tree for IoT environments security. *Journal of Reliable Intelligent Environments*, jul. 2022. ISSN 2199-4676. Disponível em: ⟨https://doi.org/10.1007/s40860-022-00184-3⟩.

[17] UMAR, M. A. et al. Effects of feature selection and normalization on network intrusion detection. *Data Science and Management*, v. 8, n. 1, p. 23–39, mar. 2025. ISSN 2666-7649. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S2666764924000390⟩.

[18] ABDALLAH, E. E.; ELEISAH, W.; OTOOM, A. F. Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey. *The 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40)*, v. 201, p. 205–212, jan. 2022. ISSN 1877-0509. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1877050922004422⟩.

[19] RAMACHANDRAN, S. et al. A survey on supervised machine learning algorithms. *AIP Conference Proceedings*, v. 3279, n. 1, p. 020003, abr. 2025. ISSN 0094-243X. Disponível em: ⟨https://doi.org/10.1063/5.0263126⟩.

[20] M. Agoramoorthy et al. An Analysis of Signature-Based Components in Hybrid Intrusion Detection Systems. In: *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)*. [S.l.: s.n.], 2023. p. 1–5. Journal Abbreviation: 2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS).

[21] THAKKAR, A.; LOHIYA, R. A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artificial Intelligence Review*, v. 55, n. 1, p. 453–563, jan. 2022. ISSN 1573-7462. Disponível em: ⟨https://doi.org/10.1007/s10462-021-10037-9⟩.

[22] Van Hauser. *The Hacker Choice's IPv6 Attack Toolkit*. 2020. Disponível em: ⟨https://github.com/vanhauser-thc/thc-ipv6/tree/v3.8⟩.

[23] Wireshark Foundation. *Wireshark*. 2025. Disponível em: ⟨https://gitlab.com/wireshark/wireshark/-/tree/release-4.2?ref_type=heads⟩.

[24] Farama Foundation. *Gymnasium*. 2024. Disponível em: ⟨https://pypi.org/project/gymnasium/⟩.

[25] CHOU, D.; JIANG, M. A Survey on Data-driven Network Intrusion Detection. *ACM Comput. Surv.*, v. 54, n. 9, out. 2021. ISSN 0360-0300. Place: New York, NY, USA Publisher: Association for Computing Machinery. Disponível em: ⟨https://doi.org/10.1145/3472753⟩.

[26] SANTOS, K. C.; MIANI, R. S.; SILVA, F. de O. Evaluating the Impact of Data Preprocessing Techniques on the Performance of Intrusion Detection Systems. *Journal of Network and Systems Management*, v. 32, n. 2, p. 36, mar. 2024. ISSN 1573-7705. Disponível em: ⟨https://doi.org/10.1007/s10922-024-09813-z⟩.

[27] E. Chatzoglou et al. Pick Quality Over Quantity: Expert Feature Selection and Data Preprocessing for 802.11 Intrusion Detection Systems. *IEEE Access*, v. 10, p. 64761–64784, 2022. ISSN 2169-3536.

[28] SAVIć, I.; LIN, X. The Analysis and Implication of Data Deduplication in Digital Forensics. In: MENG, W.; CONTI, M. (Ed.). *Cyberspace Safety and Security*. Cham: Springer International Publishing, 2022. p. 198–215. ISBN 978-3-030-94029-4.

[29] N. Q. H. Othman; J. Li; Q. Yang. Multi-Agent Reinforcement Learning Aided Resource Allocation With SARSA in UAV Networks. In: *2023 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*. [S.l.: s.n.], 2023. p. 1–6. ISBN 2837-116X. Journal Abbreviation: 2023 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC).

[30] I. Zaghbani; R. Jarray; S. Bouallègue. Comparative Study of Q-Learning and SARSA Algorithms for UAV Path Planning in 3D Environments. In: *2024 IEEE 28th*

International Conference on Intelligent Engineering Systems (INES). [S.l.: s.n.], 2024. p. 000245–000250. ISBN 1543-9259. Journal Abbreviation: 2024 IEEE 28th International Conference on Intelligent Engineering Systems (INES).

[31] SIVAMAYIL, K. et al. A Systematic Study on Reinforcement Learning Based Applications. *Energies*, v. 16, n. 3, 2023. ISSN 1996-1073.

[32] CHAKRABORTY, J.; MAJUMDER, S.; MENZIES, T. Bias in machine learning software: why? how? what to do? In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2021. (ESEC/FSE 2021), p. 429–440. ISBN 978-1-4503-8562-6. Event-place: Athens, Greece. Disponível em: ⟨https://doi.org/10.1145/3468264.3468537⟩.

[33] SMITH, B.; KHOJANDI, A.; VASUDEVAN, R. Bias in Reinforcement Learning: A Review in Healthcare Applications. *ACM Comput. Surv.*, v. 56, n. 2, set. 2023. ISSN 0360-0300. Place: New York, NY, USA Publisher: Association for Computing Machinery. Disponível em: ⟨https://doi.org/10.1145/3609502⟩.

[34] Q. Nguyen; N. Teku; T. Bose. Epsilon Greedy Strategy for Hyper Parameters Tuning of A Neural Network Equalizer. In: *2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA)*. [S.l.: s.n.], 2021. p. 209–212. ISBN 1849-2266. Journal Abbreviation: 2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA).

[35] BULUT, V. Optimal path planning method based on epsilon-greedy Q-learning algorithm. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, v. 44, n. 3, p. 106, mar. 2022. ISSN 1806-3691. Disponível em: ⟨https://doi.org/10.1007/s40430-022-03399-w⟩.

[36] CINCOTTA, P. M. et al. The Shannon entropy: An efficient indicator of dynamical stability. *Physica D: Nonlinear Phenomena*, v. 417, p. 132816, mar. 2021. ISSN 0167-2789. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0167278920308174⟩.

[37] ALI, A.; ANAM, S.; AHMED, M. M. Shannon Entropy in Artificial Intelligence and Its Applications Based on Information Theory. *Journal of Applied and Emerging Sciences; Vol 13, No 1 (2023)*, 2023. Disponível em: ⟨https://journal.buitms.edu.pk/j/index.php/bj/article/view/549⟩.

[38] SARAIVA, P. On Shannon entropy and its applications. *Kuwait Journal of Science*, v. 50, n. 3, p. 194–199, jul. 2023. ISSN 2307-4108. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S2307410823000433⟩.

[39] S. Dutta; A. Arunachalam; S. Misailovic. To Seed or Not to Seed? An Empirical Analysis of Usage of Seeds for Testing in Machine Learning Projects. In: *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. [S.l.: s.n.], 2022. p. 151–161. ISBN 2159-4848. Journal Abbreviation: 2022 IEEE Conference on Software Testing, Verification and Validation (ICST).

[40] AHMED, H.; LOFSTEAD, J. Managing Randomness to Enable Reproducible Machine Learning. In: *Proceedings of the 5th International Workshop on Practical Reproducible Evaluation of Computer Systems*. New York, NY, USA: Association for Computing Machinery, 2022. (P-RECS '22), p. 15–20. ISBN 978-1-4503-9313-3. Event-place: Minneapolis, MN, USA. Disponível em: ⟨https://doi.org/10.1145/3526062.3536353⟩.

[41] KACZMARCZYK, K.; MIAŁKOWSKA, K. Backtesting comparison of machine learning algorithms with different random seed. *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 26th International Conference KES2022*, v. 207, p. 1901–1910, jan. 2022. ISSN 1877-0509. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1877050922011346⟩.