To the Graduate Council:
I am submitting herewith a dissertation written by Curtis Ray Rookard Jr. entitled "Cyber Threat Detection using Multifaceted Machine Learning Approaches." I have examined the final paper copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial Engineering.

Anahita Khojandi, Major Professor

We have read this dissertation
and recommend its acceptance:

Catalin Roman

Hugh Medal

Zhongshun Shi

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:
I am submitting herewith a dissertation written by Curtis Ray Rookard Jr. entitled "Cyber Threat Detection using Multifaceted Machine Learning Approaches." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial Engineering.

Anahita Khojandi, Major Professor

We have read this dissertation
and recommend its acceptance:

Catalin Roman

Hugh Medal

Zhongshun Shi

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Cyber Threat Detection using Multifaceted Machine Learning Approaches

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Curtis Ray Rookard Jr.

December 2024

To my mom and my wife who always pushed me to further my education

Acknowledgements

I would like to extend my utmost appreciation and thanks to my advisor, Dr. Anahita Khojandi, who guided me through my studies at the University of Tennessee. Without her support, I would not have been able to complete my dissertation and become the researcher I am today. I would also like to extend a warm thanks to my committee members, Drs. Catalin Roman, Hugh Medal, and Zhongshun Shi for their willingness to serve on my committee and for their comments that made this work better. Additionally, I couldn't have completed this work without the support of my mom and dad. Thank you for always encouraging me to continue my education. Finally, I would like to thank my wife, Lily, for being there for me through all the late nights during my homework and research. I appreciate all you do.

The research presented in this dissertation is partially supported by the Logistics Management Institute-Department of Defense Science, Mathematics, and Research for Transformation (SMART) scholarship program with contract number W911NF-22-2-0156. Materials from chapter 1 have previously appeared in [98], in the Elsevier journal *Computers & Security*, Vol. 140, C. Rookard and A. Khojandi, "Recurrent Reinforcement Learning for Cyber Threat Detection on IoT Devices," Elsevier, available online at https://doi.org/10.1016/j.cose.2024.103786.

Materials from chapter 2 are from [99], ©2024 IEEE. Reprinted, with permission, from C. Rookard and A. Khojandi, "Unsupervised Machine Learning for Cybersecurity Anomaly Detection in Traditional and Software-Defined Networking Environments," in *IEEE Transactions on Network and Service Management*, doi:

10.1109/TNSM.2024.3490181. In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of the University of Tennessee's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Micro films and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation. Materials in chapter 3 are currently submitted and under review.

"The roots of education are bitter, but the fruit is sweet." - Aristotle

Abstract

Due to the worldwide growth of cyber threats, cybersecurity has become a field of increasing importance within the past years. Henceforth, it is imperative to design robust threat detection approaches that can identify and mitigate malicious internet traffic. Several machine learning techniques have shown immense potential for cyber threat detection.

First, we propose RRIoT, a Deep Deterministic Policy Gradient reinforcement learning (RL) algorithm in conjunction with an LSTM layer within an adversarial environment to detect and identify sequential attacks. We evaluate our method against novel and state-of-the-art ML/RL algorithms. Our results indicate that our proposed RRIoT generally performs better than existing ML algorithms and performs as well as or better than novel RL algorithms with new network architectures. We leverage Shapley Additive Global Importance (SAGE) to provide additional insight into which features contribute most to a model's performance.

Second, we present several unsupervised/semi-supervised machine learning models to combat prolific anomalous data on a computer network. Specifically, we employ five unsupervised machine learning models. We use these models separately and, when applicable, combined together to examine their anomaly detection performance on four networking datasets within both a software-defined networking and industrial internet-of-things environment. We investigate the generalizability of the models across two datasets. Our results suggest the Isolation Forest and Deep Belief Network

algorithms perform better than other models in both traditional and software-defined networking environments.

Third, we provide more robust algorithms for network data analysis through the usage of federated learning, a decentralized machine learning technique which provides a secure way to analyze data. In federated learning, clients train models on their own data and provide updates to a centralized server for creation of a global model, providing confidentiality. However, federated learning averaging algorithms do not consider underlying client health or performance. In this study, we present our novel federated learning framework, FLOWS, to alleviate this issue. FLOWS is formulated as a partially observable Markov decision process which is solved using reinforcement learning. We apply our framework for intrusion detection. FLOWS generally outperforms federated learning and improves the global model's performance.

Table of Contents

1 Recurrent Reinforcement Learning for Cyber Threat Detection on IoT Devices **1**

1.1 Introduction . . . 1

1.2 Literature Review . . . 4

1.3 Proposed Solution . . . 9

1.4 Numerical Results . . . 15

1.4.1 Data . . . 15

1.4.2 Model Hyperparameters . . . 19

1.4.3 Model Evaluation . . . 19

1.4.4 Analytical Approach . . . 21

1.4.5 Analysis I . . . 23

1.4.6 Analysis II . . . 28

1.4.7 Analysis III . . . 33

1.5 Discussion . . . 36

1.5.1 SAGE Analysis . . . 37

1.5.2 Ablation Study . . . 44

1.5.3 Intermittent Traffic . . . 47

1.5.4 Considerations for Direct Deployment to IoT and Embedded Devices . . . 49

1.6 Conclusion and Future Works . . . 52

2 Unsupervised Machine Learning for Cybersecurity Anomaly Detection in Traditional and Software-Defined Networking Environments 54

2.1 Introduction . . . 54

2.2 Literature Review . . . 59

2.3 Methods . . . 64

2.3.1 Data . . . 64

2.3.2 Models . . . 68

2.3.3 Experimental Design . . . 76

2.3.4 Model Evaluation . . . 77

2.3.5 Model Performance Metrics . . . 78

2.3.6 Identification of the Best Model . . . 80

2.3.7 Hyperparameter Analysis and Selection . . . 80

2.4 Numerical Results . . . 82

2.4.1 Experiment W-I . . . 82

2.4.2 Experiment W-II . . . 84

2.4.3 Experiment W-III . . . 84

2.4.4 Experiment A-I . . . 87

2.4.5 Experiment A-II . . . 87

2.4.6 GANomaly Performance . . . 89

2.5 Conclusion and Future Works . . . 91

3 FLOWS: Federated Learning Optimal Weighting of Sensors for Network Intrusion Detection Systems 94

3.1 Literature Review . . . 96

3.1.1 Federated Learning for Network Intrusion Detection . . . 97

3.1.2 Optimal Client Selection and Weighting . . . 97

3.1.3 Prescriptive Methods for Classification Tasks . . . 99

3.2 Methodology . . . 100

3.2.1 NSL-KDD Dataset . . . 100

3.2.2 Federated Learning . . . 103
3.2.3 POMDP Formulation . . . 104
3.2.4 Deep Recurrent Q-Network . . . 107
3.2.5 Experimental Design . . . 109
3.3 Experimental Results . . . 109
3.3.1 Performance Metrics . . . 109
3.3.2 Performance . . . 110
3.4 Discussion . . . 122
3.5 Conclusion . . . 124

Bibliography **126**

Appendix **145**
A RRIoT Additional Results . . . 145
A.1 Effects of increasing testing dataset size . . . 145
A.2 SAGE Plots . . . 148
B FLOWS Additional Results . . . 150
B.1 Normal and zero-day client numerical results . . . 150
B.2 Normal and resilient client numerical results . . . 150
B.3 Normal and vulnerable client numerical results . . . 151
B.4 Effects of varying partial rewards and punishments . . . 153

Vita **155**

List of Tables

1.1 Parameters used for our RL agents and environments. 20

1.2 Results from the base models using the TON-IoT GPS dataset. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds. 24

1.3 Results from the base models using the TON-IoT Modbus dataset. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds. 26

1.4 Results from the base models using the TON-IoT Weather dataset. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds. 28

1.5 Results from the base models compared with the recurrent models using the TON-IoT GPS dataset. The recurrent models are located on the leftmost part of the table. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds. . 31

1.6 Results from the base models compared with the recurrent models using the TON-IoT Modbus dataset. The recurrent models are located on the leftmost part of the table. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds. . 32

1.7 Results from the base models compared with the recurrent models using the TON-IoT Weather dataset. The recurrent models are located on the leftmost part of the table. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds. . 33
1.8 Results from the base models using the TON-IoT GPS dataset wherein a single feature, in this instance, latitude, withheld from the model. Statistically significant results are indicated by an asterisk (*). 46
1.9 Results from the base models using the TON-IoT Modbus dataset wherein a single feature, in this instance, the first input register, withheld from the model. Statistically significant results are indicated by an asterisk (*). 47
1.10 Results from the base models using the TON-IoT Weather dataset wherein a single feature, in this instance, humidity, withheld from the model. Statistically significant results are indicated by an asterisk (*). 48
1.11 A sample of observations of the intermittent dataset for the GPS telemetry data 48
1.12 Results from the base models using the TON-IoT GPS dataset wherein specific observations are incomplete. Statistically significant results are indicated by an asterisk (*). 48
1.13 Results from the base models using the TON-IoT Modbus dataset wherein specific observations are incomplete. Statistically significant results are indicated by an asterisk (*). 50
1.14 Results from the base models using the TON-IoT Weather dataset wherein specific observations are incomplete. Statistically significant results are indicated by an asterisk (*). 50

2.1 An overview of selected literature concerning the application of ML towards network intrusion detection (OC)SVM: (One Class) Support Vector Machine, RBM: Restricted Boltzmann Machine, DBN: Deep Belief Network. 63
2.2 Summary of key hyperparameters in our unsupervised learning models. 82
2.3 Experiment W-I: The mean and standard deviation for the NSL-KDD, KDD-Cup 99, and CIC-IDS2017 Datasets. Statistical significance at 5% is indicated by an asterisk (*). 83
2.4 Experiment W-II: The mean and standard deviation for the NSL-KDD and KDD-Cup 99 dataset using a subset of features present in SDN traffic alongside the DNP3 dataset. Statistical significance at 5% is indicated by an asterisk (*). 85
2.5 Experiment W-III: The mean and standard deviation for the NSL-KDD, KDD-Cup 99, and DNP3 Datasets models paired with the Restricted Boltzmann Machine. Statistical significance at 5% is indicated by an asterisk (*). 86
2.6 Experiment A-I: The mean and standard deviation for training and testing across different datasets to determine generalizability of our models. Statistical significance at 5% is indicated by an asterisk (*). . 88
2.7 Experiment A-II: The mean and standard deviation for training and testing across different datasets to determine generalizability of our models when combined with the RBM. Statistical significance at 5% is indicated by an asterisk (*). 88

2.8 Summary of performance comparison between GAN and other models presented in the study, where GAN outperformed other models in certain metrics (in bold). Asterisks indicate statistically significant results at an alpha of 5%. The GAN-based model was unable to outperform the other models when trained and tested on the instances of full, SDN-feature, and RBM with feature extraction on the NSL-KDD dataset. 90

3.1 An overview of related literature and attributes of each work sorted chronologically. 96

3.2 An overview of both system and client symbols present in our POMDP. 107

3.3 Results from the benchmark federated learning and our FLOWS framework models using two clients on the NSL-KDD intrusion detection dataset. Statistically significant results are indicated by an asterisk (*). 111

3.4 Results from the benchmark federated learning and our FLOWS framework models using three clients on the NSL-KDD intrusion detection dataset. Statistically significant results are indicated by an asterisk (*). 112

3.5 Results from the benchmark federated learning and our FLOWS framework models using four clients on the NSL-KDD intrusion detection dataset. Statistically significant results are indicated by an asterisk (*). 112

3.6 Mathematical TPMs and the corresponding clients in realistic networking settings. 114

3.7 Results from the benchmark federated learning and our FLOWS framework using a sensitivity analysis wherein underlying client states are resistant to change for two clients on the NSL-KDD intrusion detection dataset. We compare our sensitivity analysis to the benchmark federated learning model. Statistically significant results are indicated by an asterisk (*). 117
3.8 Results from the benchmark federated learning and our FLOWS framework using a sensitivity analysis wherein one client has a higher probability of being attacked and staying attacked while the other client has a higher probability of transitioning to normal and staying in the normal state for two clients on the NSL-KDD intrusion detection dataset. We compare our sensitivity analysis to the benchmark federated learning model. Statistically significant results are indicated by an asterisk (*). 119
3.9 Results from the benchmark federated learning and our FLOWS framework using a sensitivity analysis with partial rewards for partially correct classification for two clients on the NSL-KDD intrusion detection dataset. We compare our sensitivity analysis to the benchmark federated learning model. Statistically significant results are indicated by an asterisk (*). 121
A.1 Results from the base models using the TON-IoT GPS dataset with a larger test dataset size. Statistically significant results are indicated by an asterisk (*). 146
A.2 Results from the base models using the TON-IoT Modbus dataset with a larger test dataset size. Statistically significant results are indicated by an asterisk (*). 146
A.3 Results from the base models using the TON-IoT Weather dataset with a larger test dataset size. Statistically significant results are indicated by an asterisk (*). 146

B.1 Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset with one client having a normal TPM and the other client having a zero-day TPM. Statistically significant results are indicated by an asterisk (*). 151
B.2 Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset with one client having a normal TPM and the other client having a resilient TPM. Statistically significant results are indicated by an asterisk (*). 152
B.3 Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset with one client having a normal TPM and the other client having a vulnerable TPM. Statistically significant results are indicated by an asterisk (*). 152
B.4 Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset for varying rewards for full correct, partial, and incorrect classification, respectively. Statistically significant results are indicated by an asterisk (*) 154

List of Figures

1.1 An overview of the process taken in RL, in which an intelligent agent interacts with an environment through an action A_t given a state S_t. The agent then obtains a reward R_t, and a new state is reached. . . . 6

1.2 Overview of RRIoT incorporating an adversarial environment. Both the environment and classifier RL agents are recurrent networks. Steps are labeled A through E. In step A, the environment agent obtains a sample from the TON-IoT device dataset. Step B involves the classifier agent determining the traffic class based upon the selected sample from the environment agent. In step C, the rewards are calculated for both the environment agent and the classifier agent based upon their performance. The rewards are then provided to the Q_e-function and Q_c-function for both the environment and classifier agent in step D, which after the reward update, are then sent back to the agents in step E. 12

1.3 Distribution of the data present in the TON-IoT GPS telemetry dataset. 18

1.4 Distribution of the data present in the TON-IoT Modbus telemetry dataset. 20

1.5 Distribution of the data present in the TON-IoT Weather telemetry dataset. 20

1.6 Bar chart of correctly identified, false positive, and false negative for the AE-Dueling DQN algorithm on the TON-IoT GPS dataset. . . . 25

1.7 Confusion Matrix for the AE-Dueling DQN algorithm on the TON-IoT GPS dataset. 25
1.8 Bar chart of correctly identified, false positive, and false negative for the base DQN algorithm on the TON-IoT Modbus dataset. 27
1.9 Confusion Matrix for the base DQN algorithm on the TON-IoT Modbus dataset. 27
1.10 Bar chart of correctly identified, false positive, and false negative for RIoT on the TON-IoT weather dataset. 29
1.11 Confusion Matrix for RIoT on the TON-IoT weather dataset 29
1.12 Bar chart of correctly identified, false positive, and false negative for RRIoT on the TON-IoT weather dataset. 34
1.13 Confusion Matrix for RRIoT on the TON-IoT weather dataset. . . . 34
1.14 SAGE Plot for the AE-Dueling DQN Algorithm for Analysis I using the GPS telemetry dataset. 38
1.15 SAGE Plot for the AE-RL Algorithm for Analysis I using the GPS telemetry dataset. 38
1.16 SAGE Plot for the AE-Dueling DQN Algorithm for Analysis I using the Modbus telemetry dataset. 39
1.17 SAGE Plot for RIoT for Analysis I using the Modbus telemetry dataset. 39
1.18 SAGE Plot for RIoT for Analysis I using the weather telemetry dataset. 41
1.19 SAGE Plot for the AE-Dueling DQN Algorithm for Analysis I using the weather telemetry dataset. 41
1.20 SAGE Plot for the AE-Dueling DRQN Algorithm for Analysis II using the GPS telemetry dataset. 42
1.21 SAGE Plot for the AE-DRQN Algorithm for Analysis II using the GPS telemetry dataset. 42
1.22 SAGE Plot for the AE-Dueling DRQN Algorithm for Analysis II using the Modbus telemetry dataset. 43

1.23 SAGE Plot for RRIoT for Analysis II using the Modbus telemetry dataset. 43
1.24 SAGE Plot for RRIoT-AC for Analysis II using the Weather telemetry dataset. 45
1.25 SAGE Plot for RRIoT-A for Analysis II using the Weather telemetry dataset. 45

2.1 Traditional Networking Architecture versus Software Defined Networking [70]. 56
2.2 Traditional GAN Architecture, adapted from [4]. 70
2.3 GAN-based Anomaly Detection Architecture, adapted from [4]. . . . 71
2.4 Training procedure versus testing procedure of our GAN-based model. 79

3.1 Our novel FLOWS framework which comprises of two primary stages. Stage one consists of obtaining observations from our first stage of our federated learning model. These observations are then passed to the second stage of our model which are then used to update the belief of the edge clients and the full system. Our RL agent then takes an action which corresponds to a weighting schema for normal and attacked nodes in the system. These new weights are then passed back to an updated federated averaging algorithm. 101
3.2 Distribution of the traffic present in the NSL-KDD dataset. 102
3.3 Timeline of our POMDP execution over federated learning rounds. . . 104
3.4 DRQN within the context of a POMDP. The DRQN inputs observations from the environment as the true underlying state is not fully observable and outputs actions for interaction with the environment. 108
3.5 Experimental design for our FLOWS framework. 110
3.6 Belief vs underlying state, DRQN action, and cumulative accuracy and G-Mean comparison graph for our sensitivity analysis wherein clients are resistant to changing states. 116

3.7 Belief vs underlying state, DRQN action, and cumulative accuracy and G-Mean comparison graph for our sensitivity analysis wherein one client is more likely to be vulnerable to attacks and remain attacked and the other client is more likely to be resilient to attacks and remain in the normal state. 118
3.8 Comparison of RL agent reward performance with a normal rewards structure (left) versus modified reward structure (right). While the RL agent starts at a higher reward level with the modified reward structure, the cumulative growth in rewards is lower. 121
3.9 Comparison of RL agent action with a normal rewards structure (left) versus modified reward structure (right). While the RL agent with normal rewards tends to select a variety of actions, the RL agent with modified rewards selects only one. 121
A.1 Analysis I SAGE Plots presented for the GPS telemetry dataset on the AE-RL, AE-Dueling DQN, and RIoT Models, respectively. 148
A.2 Analysis I SAGE Plots presented for the Modbus telemetry dataset on the AE-RL, AE-Dueling DQN, and RIoT Models, respectively. 148
A.3 Analysis I SAGE Plots presented for the weather telemetry dataset on the AE-RL, AE-Dueling DQN, and RIoT Models, respectively. 148
A.4 Analysis II SAGE Plots presented for the GPS telemetry dataset on the AE-DRQN, AE-Dueling DRQN, and RRIoT Models (RRIoT-A, RRIoT-C, and RRIoT-AC), respectively. 149
A.5 Analysis II SAGE Plots presented for the Modbus telemetry dataset on the AE-DRQN, AE-Dueling DRQN, and RRIoT Models (RRIoT-A, RRIoT-C, and RRIoT-AC), respectively. 149
A.6 Analysis II SAGE Plots presented for the weather telemetry dataset on the AE-DRQN, AE-Dueling DRQN, and RRIoT Models (RRIoT-A, RRIoT-C, and RRIoT-AC), respectively. 149

Chapter 1

Recurrent Reinforcement Learning for Cyber Threat Detection on IoT Devices

1.1 Introduction

As modern society continues to incorporate the usage of computing systems, there has been a plethora of devices that have been introduced into our cyber ecosystem, specifically internet-of-things (IoT) devices. As a result of the advent of these devices into our daily lives, there are now more vulnerabilities present that can be exploited by cyber attackers. Attackers aim to subvert the confidentiality, integrity, and availability of these IoT devices, thus requiring the need for robust security controls.

Intrusion detection systems provide an additional layer of security for most computing systems or networks. When implemented, intrusion detection systems are classified as either host-based, or network-based. Host-based intrusion detection systems strictly monitor one host/computing system and can complement an already existing host-based firewall or endpoint detection and response (EDR) software. Network intrusion detection systems provide security for a network of computers.

These intrusion detection systems complement network firewalls. As opposed to firewalls, intrusion detection systems have the ability to be aided by machine learning (ML) and other artificial intelligence to prevent cyber attacks, whereas firewalls are usually static and rule-based [54].

These ML and artificial intelligence (AI) algorithms present several benefits for cybersecurity. ML algorithms can analyze vast amounts of network data in real-time, recognize patterns in network traffic and anomalies in connections, and identify potential threats prompting an alert to analysts before they pose a threat to information systems. ML and AI can additionally improve cyber incident response and enhance threat intelligence capabilities, henceforth enabling security teams to quickly detect and respond to emerging threats. As cyber threats rapidly evolve, the use of ML and AI, especially with respect to the design of smart intrusion detection systems, will become increasingly critical in safeguarding organizations' digital assets and ensuring the confidentiality, integrity, and availability of sensitive data [54].

Most intrusion detection models focus primarily on the application of ML algorithms for the detection of cyber attacks. However, to the best of our knowledge, the application of reinforcement learning (RL) towards cybersecurity, especially with respect to intrusion detection systems, is limited. As a result, the notion of providing rewards to intelligent intrusion detection systems has not been explored thoroughly in current literature. This dearth additionally fails to incorporate potentially better intelligent models not just in enterprise computer networks, but also IoT and Industrial Control Systems (ICS) networks. Henceforth, we propose a novel RL model for IoT systems, **RRIoT** (**R**ecurrent **R**einforcement learning on **IoT**), for expert system-based intrusion detection. RRIoT improves upon the adversarial environment in [16] during the model training process and also better captures the sequential nature of our IoT network traffic data through a novel recurrent network architecture. Our study proposes several major additions to the intrusion detection system literature.

- We utilize an adversarial environment with intelligent environment agent. This intelligent adversarial agent, in this instance serving as an "attacker" also applies a RL model similar to that of the "defender" agent. The environment agent works to maximize its own rewards akin to that of the defender, and thus selects attacks which maximizes its rewards.

- We introduce a recurrent layer for the deep Q-network for both the classifier (defender) and the environment (attacker) agent. We find that applying a recurrent layer improves classification of cyber attacks even when added to the environment agent. In addition we alter the Q-network architecture by also presenting a dueling recurrent architecture that splits the Q-network into two streams, the advantage and the value streams. We apply this dueling deep recurrent Q-network on both the classifier and the defender and observe improved performance.

- We introduce a novel recurrent actor-critic network using the deep deterministic policy gradient (DDPG) for both the classifier and environmental agent for intrusion detection, RRIoT. We apply recurrent layers in either our actor network, our critic network, or both. Consequently, we observe better classification of cyber attacks using a recurrent layer, especially when added to both actor and critic networks. To the best of our knowledge, our study is the first to apply this recurrent DDPG with an adversarial environment for intrusion detection on IoT systems.

In addition to furthering the intrusion detection literature by introducing our novel RIoT, we also broaden the scope of intrusion detection by applying several other novel RL benchmark algorithms towards our IoT devices, which are often not always considered in the security posture of an organization but are still susceptible to cyber attacks. We analyze the IoT device output traces, that is, the respective readings and measurements output by the device to determine whether the device is under

cyberattack. Our study is unique by instead incorporating telemetry data for IoT intrusion detection.

We provide further analysis of our model performance through the utilization of Shapley Additve Global Importance (SAGE) from [24] for our models utilizing the IoT GPS tracker, weather station, and industrial Modbus connections. SAGE determines which features are the most importance with respect to model performance and considers the global importance of dataset features. We also include ablation studies wherein we withhold certain features from our model in order to further support the importance of our model features. We then provide model performance in which IoT device telemetry data is intermittent to assess possible real-world scenarios of incomplete data.

The rest of the paper is constructed as follows: Section 1.2 provides an overview of the literature concerning intrusion detection in cybersecurity and introduces the work in Caminero et al. [16]. In section 1.3, we discuss the methodology proposed in our study, to include the RL environment, the models, and the performance metrics applied in the study. Section 1.4 provides the performance of our models and SAGE outputs. Section 1.5 provides a discussion of our results, and section 1.6 presents the conclusion and possibilities for future work.

1.2 Literature Review

Several ML and deep learning algorithms have been applied previously for computer security, especially within the field of network intrusion detection. Wang et al. [118] applied a multi-layer perceptron in order to detect distributed denial-of-service attacks on an information network. Their model selected only the optimal features that would assist with intrusion detection, specifically with respect to transmission control protocol (TCP) packets. Their study, however, primarily focused on the application of their model only on a single attack, wherein there are several different cyber attacks. In addition, Catillo et al. [17] provided a method to detection anomalous

cyber traffic in cyber-physical systems utilizing an autoencoder-based algorithm that is outlier aware. Aldwairi et al. [7] applied an unsupervised neural network, the Restricted Boltzmann Machine, for anomaly detection in NetFlow traffic. Generative adversarial networks were also employed in Kim et al. [55]; the researcher applied a generative adversarial network using autoencoders in order to detect potential zero-day exploits in malware samples. The aforementioned works focused primarily on classifying traffic as either normal or anomalous/attack traffic; while this approach is efficient at detecting attacks, it fails to provide a means for more granular classification of attack traffic. In addition, these previously mentioned works, however, primarily fail to address a potential system that detects anomalous traffic and is rewarded based upon model performance. Using this notion, we thus determine that RL would be a promising direction of research.

Cybersecurity has been assisted by RL algorithms previously in a multitude of capacities critical to information security and cybersecurity. RL, as described briefly in Figure 1.1, underscores the advantages of automating vulnerability assessments through penetration testing as demonstrated in [48]. Additionally, Ghanem and Chen [39] mechanized this process by employing a partially-observable Markov decision process (POMDP). The DQN was a cornerstone for analysis in the research by Hu et al. [48], revealing its high efficacy, especially when integrated with the open-source intelligence tool, Shodan, as highlighted by Fernández-Caramés and Fraga-Lamas [35]. Furthermore, post-breach actions were delved into by Maeda and Mimura [69], who melded RL with the renowned post-exploitation tool, Powershell Empire. This amalgamation utilized the Powershell tool alongside A2C, Q-Learning, and SARSA RL methods. While these works all demonstrate the benefit of the application of RL towards cybersecurity, several works such as those presented in the works of Hu et al. [48] and Maeda and Mimura [69], focused primarily on attack-centric activities, such as testing for vulnerabilities or post-exploitation activities. This aspect thus prompts the question instead as to how defenders can better utilize RL for cybersecurity.

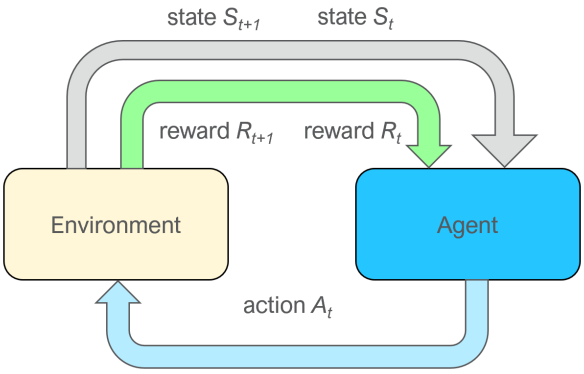


Figure 1.1: An overview of the process taken in RL, in which an intelligent agent interacts with an environment through an action A_t given a state S_t. The agent then obtains a reward R_t, and a new state is reached.

The cybersecurity workforce, especially with respect to the assignment of cyber defenders, also benefits from applications of RL. Ganesan et al. [38] combined RL with optimization methods; both an integer programming model and a dynamic optimization model analyzed the availability of cybersecurity analysts in response to cyber-attacks using a genetic algorithm. This approach primarily focused on the management aspect of cyber defense, as opposed to using RL for technical threat detection. Game theory provides an excellent opportunity to further RL. Khoury and Nassar [53] applied game theory and RL techniques within an industrial control system (ICS)/supervisory control and data acquisition environment and managed to obtain a Nash equilibrium when using multi-agent RL. Elderman et al. [32] also applied various RL techniques, including Monte Carlo learning and Q-learning, in an adversarial environment and found that the Monte Carlo learning approach was most effective for the defender. Both previous approaches were more focused on meta-defense as opposed to direct approaches to cyber defense. Deep Q-Learning with experience replay was the selected RL algorithm in Xia et al. [123]; the researchers found that their proposed RL algorithm would be more able to discern anomalies compared with other ML algorithms. Piplai et al. [88] applied Q-learning in conjunction with the application of knowledge graphs for malware samples obtained

from VirusTotal as seen from Masri and Aldwairi [71], and determined that RL was beneficial for malware identification.

Similar works previously using RL in the field of cybersecurity, specifically to detect cyber attacks, include that presented by Lopez-Martin et al. [68]. Lopez-Martin et al. [68] proposed a variety of RL models, to include DQNs, double DQNs, policy gradient and actor-critic methods across the popular NSL-KDD dataset from Tavallaee et al. [114] and the AWID dataset from Kolias et al. [58]. They find that their DDQN model had the best results against both the ML and RL algorithms applied. In their research, Dowling et al. [30] utilized RL with the state–action–reward–state–action (SARSA) technique on an active web server. This server functioned as a honeypot and cataloged cyber-attacks and defensive responses. Another study by Sethi et al. [105] integrated a Deep Q-Network strategy with an adversarial ML agent. Their objective was to craft a context-aware intrusion detection system, examining datasets like NSL-KDD, UNSW-NB15 [79], and the AWID dataset. Their findings suggested that their model's accuracy was comparable, if not superior, to several supervised ML approaches. Additionally, Caminero et al. [16] employed an adversarial deep Q-learning agent against a classifier deep Q-learning agent. Their results showcased improved metrics over other ML and RL techniques when tested on the NSL-KDD and AWID datasets. [96] proposed the usage of a DQN for intrusion detection on IoT devices and found that their proposed DQN classifier outperformed all other machine learning benchmarks. These approaches mainly capture the importance of applying RL towards the field of intrusion detection; however, none of these works focused on detecting attacks based upon telemetry data, only on network data. This gap thus necessitates the question as to the effectiveness of RL for cyber threat detection on telemetry data.

Previous works have explored IoT attack detection using ML approaches. These works have analyzed various IIoT and IoT technologies. In [34], an approach combining a CNN with an optimization algorithm was implemented for intrusion detection for IoT devices. The authors implemented a feature selection method

by applying a growth optimizer via a whale optimization algorithm. A CNN was then applied to the selected features on the KDD-Cup 99, NSL-KDD, CICIDS-2017, and Bot-IoT datasets. Additionally, in the work presented in [86] also employed a feature selection method based on optimization. In their work, an oppositional Aquila optimizer was employed in the context of an internet-of-drones environment. They applied an XGBoost classifier and determined that their proposed model outperformed other methods on the NSL-KDD and CICIDS-2017 datasets. [111] applied a feature selection method using traffic flow features for intrusion detection on botnet traffic. Their method maintained a high accuracy while also maintaining a low false positive rate. While these works do highlight the need for IoT device protection, they again only consider network traffic as opposed to detecting attacks using other pertinent sources of data.

From our survey of the literature, while we do find instances of reinforcement learning-based intrusion detection on network data, what we do find is a lack of literature concerning the implementation of these reinforcement learning algorithms on IoT data, especially IoT *telemetry* data. The usage of real-life telemetry data is limited in the present literature; however, telemetry data provides essential information for cyber defenders. The usage of telemetry data can enrich the viewpoint of cyber defenders and can assist with gaining better information about device health on their network. Additionally, this form of data serves as one of the, if not only, means of the presence of a cyber attack in certain instances [9]. To the best of our knowledge, our study is the first that applies these RL algorithms on IoT telemetry data, thus addressing a crucial gap in the IoT cyber attack detection literature.

Additionally, we further the present cyber intrusion detection literature by proposing the novel implementation of our RL algorithm, RRIoT, which applies a novel recurrent DDPG network architecture on IoT devices. RRIoT also leverages the incorporation of an adversarial environmental agent to better training for the classifier. We further provide insight into our model performance through the use of SAGE to determine the features present that contribute most to our model

performance. We include ablation studies and sensitivity studies in our work to further determine the significance of features present in our IoT telemetry dataset and confirm the performance of our RL models.

1.3 Proposed Solution

In the following subsections, we provide an overview of the novel RL model proposed in our study, in addition to the benchmark RL algorithms and the adversarial environment incorporated into RIoT. We apply the popular Q-learning algorithm for our RL algorithms. Q-learning is a popular RL algorithm that is used to learn a policy for a Markov Decision Process (MDP). The objective of Q-learning is to learn a function $Q(s, a)$, which represents the expected total reward of taking action a in state s and following the optimal policy thereafter. The Q-learning algorithm works by iteratively updating the Q-function. With an initial state s_t and initial action an action a_t and corresponding next state s_{t+1}, and next action a_{t+1}, the Q-function is approximated as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}))$$

where r_{t+1} is the reward, α denotes the learning rate, and γ serves as the discount factor that balances immediate and future rewards. In Q-learning, the Q-function is represented as a lookup table or a function approximator, such as a neural network. We will first over our novel RL model, RIoT, and its recurrent counterpart, RRIoT, and then cover the novel RL models that will be used in comparison. We will cover both the non-recurrent and recurrent novel RL benchmark models for our analysis to determine the best performance on the IoT devices.

RIoT and RRIoT

RIoT builds upon the implementation of the Deep Deterministic Policy Gradient model (DDPG). The DDPG algorithm, as proposed by Lillicrap et al. [62], is an actor-critic method that learns both a deterministic policy function $\mu(s)$ and an action-value function $Q(s, a)$ in a continuous action space. In the DDPG algorithm, the actor network learns the policy function $\mu(s)$ and the critic network learns the action-value function $Q(s, a)$.

The actor network updates the policy parameters θ_μ to maximize the expected cumulative reward:

$$\nabla_{\theta_\mu} J \approx \mathbb{E} s_t[\nabla_a Q(s_t, a)|a = \mu(s_t)\nabla_{\theta_\mu}\mu(s_t|\theta_\mu)]$$

where J is the expected cumulative reward, and s_t is the current state.

The critic network updates the action-value function parameters by minimizing the mean-squared error between the estimated action-value $Q(s, a)$ and the target value:

$$L(\theta_Q) = \mathbb{E}(s_t, a_t, r_t, s_{t+1}) \left[(y_t - Q(s_t, a_t|\theta_Q))^2\right]$$

where y_t is the target value for time step t, given by the following equation:

$$y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta_\mu)|\theta_Q)$$

where r_t is the reward obtained at time step t, and γ is the discount factor.

The target action-value $Q(s_{t+1}, \mu(s_{t+1}))$ used in the target value y_t is obtained using the target actor and target critic networks, which are copies of the actor and critic network. As opposed to the DQN, the DDPG has the distinct advantages of being both an actor-critic method and implements policy-based learning. The DDPG algorithm both provides for improved performance through stabilizing learning through the actor-critic networks and optimizes the policy — a function that maps

states to actions. These changes allow the DDPG algorithm to more dynamically interact with the environment to learn an optimal policy for better performance [62].

RIoT (**R**einforcment learning for **IoT**) has several improvements compared with the base implementation of the DDPG. First, we implement our algorithm within an adversarial environment, as shown in Figure 1.2. In this environment, one agent serves as the environment agent playing the role of an attacker, while the other agent serves as the classifier agent, synonymous with a defender agent. The environment agent and classifier agent both utilize DDPG algorithms to best select actions. Within this environment, we obtain a state $s \in S$ corresponding to a sample of IoT device traffic from our selected TON-IoT telemetry dataset; an action $a \in A$ is the classification of traffic in the set of all traffic classes. For the classifier agent, this action a_c is the classification of the traffic, while for the environment agent, this action a_e is the type of attack selected from the sample. We use rewards $r_c \in \{0, 1\}$ corresponding to incorrect and correct classification, respectively. The classifier agent earns a reward of 1 if it classifies the traffic accurately and 0 if not. Respectively for the environment agent, rewards $r_e \in \{0, 1\}$ corresponding to a reward of 1 if its selected traffic is misclassified by the classifier, else its reward is 0. In this implementation with our DDPG algorithm, there are a total of four neural networks within the RL environment - an environment agent actor network, an environment agent critic network, a classifier agent actor network and a classifier agent critic network. We also incorporate the novel addition of a recurrent, LSTM layer for both our actor and critic networks. We designate this model throughout the paper as **RRIoT** for **R**ecurrent **R**einforcement Learning for **IoT** systems. This recurrent layer aims to better account for our IoT traffic, which is *temporal* in nature. Our hypothesis for this work is that addition of these recurrent layers will provide a "memory" that considers the time-series nature of our data. We delineate whether the actor network, critic network, or both have an LSTM using the acronyms **RRIoT-A**, **RRIoT-C**, and **RRIoT-AC**.

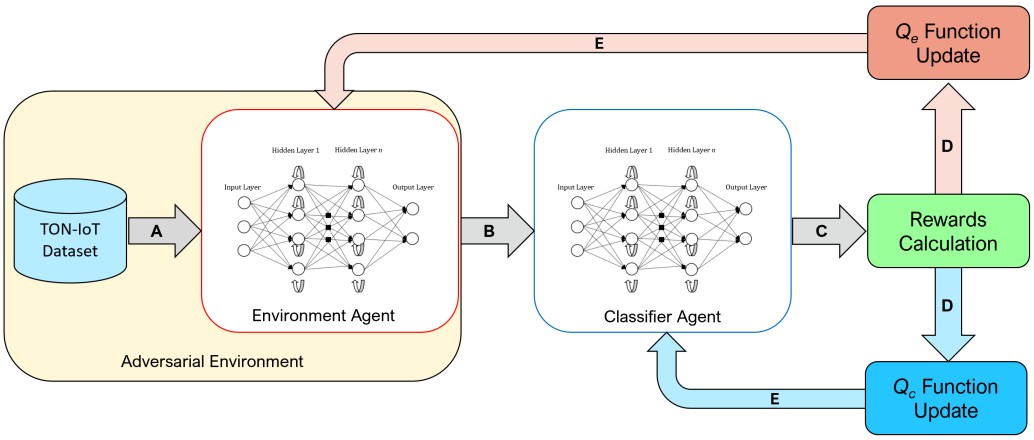


Figure 1.2: Overview of RRIoT incorporating an adversarial environment. Both the environment and classifier RL agents are recurrent networks. Steps are labeled A through E. In step A, the environment agent obtains a sample from the TON-IoT device dataset. Step B involves the classifier agent determining the traffic class based upon the selected sample from the environment agent. In step C, the rewards are calculated for both the environment agent and the classifier agent based upon their performance. The rewards are then provided to the Q_e-function and Q_c-function for both the environment and classifier agent in step D, which after the reward update, are then sent back to the agents in step E.

Novel Reinforcement Learning Benchmark Algorithms

AE-RL and Deep Recurrent Q-Networks The AE-RL algorithm was initially proposed in the work of [16]. This algorithm originally implemented the usage of an adversarial environment for reinforcement learning; that is, in addition to utilizing a DQN for intrusion detection classification, they also applied both an adversarial DQN agent which selected samples from network traffic to be classified by the classifier agent. For their implementation, they applied their model on netflow data such as the NSL-KDD and AWID dataset. We apply their model now on IoT telemetry data in this study.

In addition to the aforementioned AE-RL model, we also implement a Deep Recurrent Q-Network (DRQN) variant. DRQNs, as introduced in [43], incorporate elements of both the aforementioned Deep Q-Networks as presented in [76] and recurrent neural networks as described in [73]. Instead of constructing a DQN using

only dense layers, an long-short term memory (LSTM) layer is instead added to the network architecture. This change in architecture provides several benefits. The recurrent layers also provides more efficient learning of time-series data as presented commonly with computer network traffic. We thus refer to this extension of the base AE-RL model as the *AE-DRQN* model in this study.

Deep Dueling Q-Networks and Deep Dueling Recurrent Q-Networks Another popular extension of the DQN, as proposed by [120], involves a dueling architecture in which there are two estimators, one for the value and advantage functions. For the dueling DQN, the Q-function is split into two streams: one for the value function and one for the advantage function. The value stream estimates the value of being in a given state, while the advantage stream estimates the advantages of each possible action relative to the value of the state.

The advantage stream is defined as the difference between the estimated value of the state and the estimated value of each action in that state. This separation allows the Dueling DQN to better generalize across various actions. With this change, the equation for the calculation of the Q values henceforth transforms into:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a_{t+1} \in \mathcal{A}} A(s, a_{t+1})$$

where $V(s)$ is the state value function, $A(s,a)$ is the action advantage function for action a in state s, and $\mathcal{A}$ is the set of all possible actions, in this case, the predicted types of network traffic. During training, the dueling DQN is updated using the same Q-learning algorithm as the standard DQN. The state value function $V(s)$ estimates the value of being in a particular state regardless of the action taken, and is calculated through the following equation:

$$V(s) = \mathbb{E}_{a \sim \pi}[Q(s,a)]$$

where π is the policy, and the action advantage function $A(s, a)$ estimates the advantage of each possible action in that state relative to the value of the state, and is computed as:

$$A(s, a) = Q(s, a) - V(s)$$

The final output of the network is obtained by combining the estimated value and advantage streams. The primary advantage of the dueling DQN as opposed to a vanilla DQN is the ability of learning important states without having to learn the effect of each action; this architectural change learns better actions and tends to better stability and performance [120].

For our study, we incorporate the adversarial environment agent akin to the AE-RL and AE-DRQN models. Both our environmental agent and classifier agent implement the usage of the Dueling DQN. As such, we refer to this extension of the Dueling DQN model as the *AE-Dueling DQN* model in this study. In addition we also include an LSTM model in our AE-Dueling DQN before the split into two streams. This novel model thus combines the adversarial environment agent with recurrent layers on the Dueling DQN model. This novel model we refer to as the *AE-Dueling DRQN* throughout this study.

Machine Learning Benchmark Algorithms

We also opt to provide three benchmark ML models, the support vector machine (SVM) with radial basis function, Naive Bayes classifier (NB), and multi-layer perceptron (MLP) to compare against our RL models.

Shapley Additive Global Importance

Oftentimes an understanding of a feature's global importance is worthwhile to determine its effect across an entire dataset. SAGE henceforth provides a method to determine a feature's global importance. SAGE values can be calculated by the

following equation:

$$\phi_i(v_f) = \frac{1}{d} \sum_{S \subseteq D \setminus \{i\}} \binom{(d-1)}{|S|}^{-1} [I(Y; X_i | X_S)]$$

where d is a singular feature from in the total feature space D, S is the subset of features from D and I is mutual information for random variables Y and X. As opposed to SHAP, which contributes to understanding model prediction, SAGE contributes to understanding model performance. [24]. We provide SAGE outputs for each of our RL models to better under each feature's global importance.

1.4 Numerical Results

1.4.1 Data

We have opted to analyze an intrusion detection dataset that is specifically focused on IoT devices. Henceforth, we analyze the TON-IoT telemetry dataset as proposed initially in [9] and expanded further in [14, 80, 81]. [9] proposed the creation of this dataset through the usage of a cyber range with various virtual IoT systems. These IoT systems include a GPS tracker, weather station, smart fridge, motion activated lights, garage lights, thermostat, and industrial Modbus connections. Virtual machines loaded with the Kali Linux operating system [44] conducted the attacks on the IoT devices and the services running on those IoT devices.. Each IoT device has their own respective features; features reflected for the GPS tracker such as latitude and longitude would be absent for the industrial Modbus connection. We analyze the GPS tracker, weather station, and industrial Modbus connection due to its prevalence in several industrial settings, applicability for real-time monitoring, and proliferate usage of sensors for measurement.

Dataset Attacks: Cyber attacks primarily fall within one or more of three categories: confidentiality attacks, integrity attacks, and availability attacks. These

three categories create the cybersecurity triad. Confidentiality ensures that only authorized individuals and processes and access and view certain data. The integrity principle states that data cannot be altered by unauthorized sources. The final principle, availability, ensures that data remains accessible. Attackers attempt to subvert the triad by either disclosing sensitive data, altering the integrity of the information, or denying the access of data. Respectively, the TON-IoT dataset has seven types of attacks that affect each of principle of the information security triad. These attacks were implemented against the various services running on the IoT devices through a suite of utilities [9]. A breakdown of each attack, separated by the security principle affected, is described as follows:

- *Confidentiality Attacks:*
  - Scanning: Oftentimes the initial action taken in cyber attacks, scanning attacks involve determining the open ports and active IP address on a network or information system. Scanning often preempts more sophisticated attacks as the attacker gains detailed knowledge of a target network or system. For the TON-IoT dataset, scanning attacks against the IoT devices involved the utilization of the Nmap [84] or Nessus [10] utility for port scanning.
  - Password: Password attacks fall primarily into either password guessing attacks, password spraying attacks, or password cracking attacks. Password guessing attacks involve several passwords being tested against one account, password spraying involves testing one password against several accounts, and password cracking involves determining the cleartext password from a cryptographic hash of a password. The Hydra utility was applied for brute-force password cracking attacks while the CeWL toolkit, both referenced in [51], was applied for dictionary password cracking attacks.

- Injection: As specified in [9], the TON-IoT dataset applies structured query language (SQL) injection attacks against the emulated IoT systems. SQL injection attacks involve exploiting the query to the dataset through supplying special characters, such as dashes, apostrophes, or similar characters. These attacks on the TON-IoT devices were conducted through the usage of several bash scripts against interactive webservers running on the IoT devices.

- *Integrity Attacks:*

  - Backdoor: Attacks involving subversion of in place authentication measures are backdoor attacks. Oftentimes, backdoors are either left by software developers either inadvertently or as a way to gain future access to an application, but can be exploited by attackers in a way to establish a persistence mechanism. The Metasploit framework [116] provided the basis for the backdoor attacks against these IoT devices.

  - XSS: Cross-Site Scripting (XSS) attacks target victims through infected websites. Attackers commonly implant malicious code on a trusted host; in several instances, these hosts are popular websites. Whenever users navigate to those websites, the malicious code is executed on their computer. Bash scripts were executed against the IoT device webservices in order to conduct XSS attacks.

- *Availability Attacks:*

  - Ransomware: Arguably one of the more popularized and damaging attacks, ransomware prevents access to files through maliciously encrypting user files. Users are often presented with a banner demanding ransom payment in return for a decryption key. While the files are altered as part of the process, the user ultimately loses access to their files. The Metasploitable

framework once again provided the basis of the Ransomware attacks against our selected IoT devices.

– DDoS: Distributed Denial of Service (DDoS) attacks utilize several infected hosts to overwhelm a victim, rendering the victim host unresponsive or causing the application to crash. These attack employ the usage of a botnet, a collection of infected machines, to launch the attack. The Python packet-crafting utility ScaPy, as described in [95], was utilized for the DDoS attacks.

Dataset Distributions: The following section describes the distributions of our selected IoT devices extracted from the TON-IoT dataset.

Figure 1.3 provides an overview of the dataset distribution for the TON-IoT GPS telemetry dataset. This dataset provides a realistic, unbalanced representation of data with respect to both normal traffic and attack traffic distribution. The dataset has 35,000 normal observations. Additionally there are 5,000 Backdoor, DDoS, Password, and Injection attacks. There are 2,833 observations corresponding to ransomware attack, and there are 577 and 500 observations for XSS and scanning

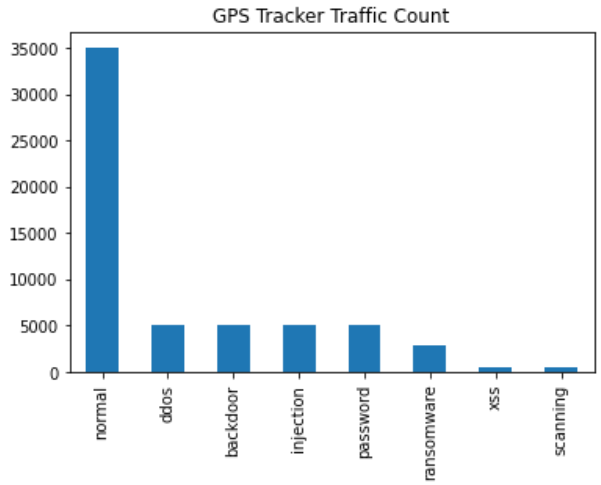


Figure 1.3: Distribution of the data present in the TON-IoT GPS telemetry dataset.

attacks, respectively [9], indicating that our dataset has a greater number of certain attacks compared with XSS and scanning.

Figure 1.4 provides an overview of the dataset distribution for the TON-IoT Modbus telemetry dataset. Unlike the GPS telemetry dataset, the Modbus telemetry dataset does not include ransomware or DDoS attacks. Akin to the GPS dataset, there are 35,000 normal observations. There are 5,000 Backdoor, Password, and Injection attacks. There are 577 observations corresponding to XSS, and there are 529 attacks corresponding to scanning attacks [9].

Figure 1.5 provides an overview of the dataset distribution for the TON-IoT weather telemetry dataset. This also dataset provides a realistic, unbalanced representation of data with respect to both normal traffic and attack traffic distribution. Distribution for the attack types is similar to the GPS telemetry dataset. The dataset has 35,000 normal observations with 5,000 Backdoor, DDoS, Password, and Injection attacks. There are 2,865 observations corresponding to ransomware attack, with 866 and 529 observations for XSS and scanning attacks, respectively [9].

1.4.2 Model Hyperparameters

For our study, we partitioned our dataset, following an allocation of 80% of observations for training purposes and the remaining 20% of observations for testing. We employed a ten-fold cross-validation technique to further assess the performance of our models. Our model hyperparameters are described briefly in Table 1.1.

1.4.3 Model Evaluation

We apply the performance metrics of accuracy, precision, recall, F1-score, and Geometric Mean (G-mean) in order to assess the performance of our models. Accuracy is calculated as $\frac{TP+TN}{TP+TN+FP+FN}$, precision is calculated as $\frac{TP}{TP+FP}$, recall is calculated as $\frac{TP}{TP+FN}$, F1-Score is calculated as $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$, and G-Mean is calculated as $\sqrt{\frac{TP}{TP+FN}\frac{TN}{FP+TN}}$. In these equations, true-positive (TP) correctly

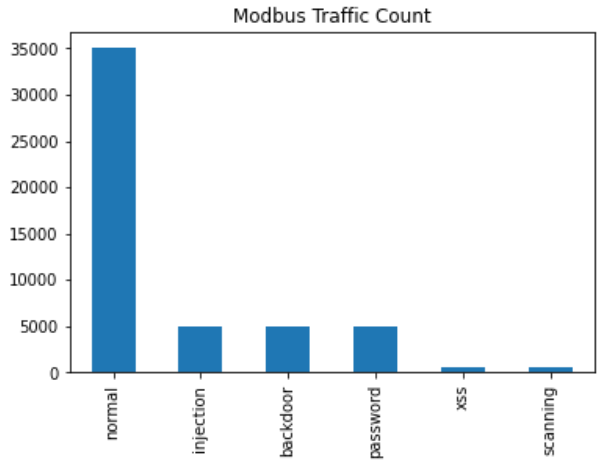


Figure 1.4: Distribution of the data present in the TON-IoT Modbus telemetry dataset.

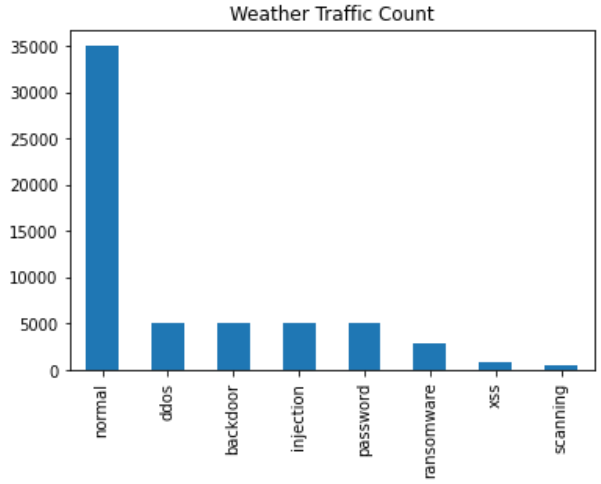


Figure 1.5: Distribution of the data present in the TON-IoT Weather telemetry dataset.

Table 1.1: Parameters used for our RL agents and environments.

Hyperparameter	**Value**	**Brief Description**
Episodes	100	Number of episodes generated
Epochs per Episode	100	Number of epochs per episode
Defender Learning Rate	0.1	Learning rate for the classifier agent
Attacker Learning Rate	0.2	Learning rate for the environment agent
Hidden Nodes	100	Number of nodes in each hidden layer
Discount Factor, γ	0.1	Reflects present value of future rewards

identifies anomalous traffic as anomalous, while false-positive (FP) incorrectly identifies non-anomalous traffic as anomalous. True-negative (TN) correctly identifies non-anomalous traffic as non-anomalous, while false-negative (FN) incorrectly identifies anomalous traffic as non-anomalous.

Accuracy is not always the best metric for evaluating anomaly detection models, as there may be fewer anomalous samples than non-anomalous ones. This distribution could potentially bias the accuracy metric towards non-anomalous samples. To address this issue, we also consider other metrics such as precision, recall, F1-score, and G-mean. The F1-score considers both precision and recall, while the G-mean measures a model's ability to balance false positives and false negatives. We intend to use the G-mean to determine the most effective model. In order to determine the best model, we compare the two best performances across the models and conduct a t-test to determine the statistical significance of the results at an α of 5%.

The Python Sci-kit learn library [85] serves as the foundation for our benchmark ML models, while our RL algorithm is programmed using Tensorflow [2]. Our experiments are performed on an Intel core i7 processor with 32GB of RAM.

1.4.4 Analytical Approach

For our study, we conduct three primary sets of analyses. Analysis I will detail the effectiveness of the non-recurrent RIoT proposed and determine its efficacy versus machine learning models and the DQN serving as benchmarks. Next, analysis II will then assess the effectiveness of adding a recurrent layer to our RL models to determine the benefit in performance. Each of our analyses are conducted by feeding in the telemetry data of each IoT device into our benchmark and novel models. The environment agent in our RL models, to include our RIoT RRIoT, takes in the telemetry data and selects a sample for our classifier agent. We provide the performance of our classifier for each of our analysis methods. Analysis III will

then leverage SAGE to determine the most important features based on model performance. Our analytical approach is described further below:

- **Analysis I**: Analysis I focuses primarily on our base RIoT, and compares the model against the aforementioned ML and RL benchmarks, to include the base DQN, the original AE-RL algorithm, and the novel benchmark algorithm with a dueling DQN (AE-Dueling DQN). We also compare RIoT with other popular ML algorithms. We hypothesize that our RL algorithms will perform better than the presented ML benchmarks. We split our analysis into three experiments using each individual dataset; these experiments are listed below:
  - *Experiment I-GPS*
  - *Experiment I-Modbus*
  - *Experiment I-Weather*
- **Analysis II**: Analysis II focuses instead on the usage of a recurrent layer in RIoT (RRIoT) on the actor, critic, and both actor and critic networks. We compare our model performance instead against the base, non-recurrent RIoT and RL algorithms and also provide novel RL benchmarks with an LSTM incorporated into their network architecture. We hypothesize that our recurrent models will perform better than their non-recurrent counterparts. We also split our analysis into three experiments using each individual dataset as delineated below:
  - *Experiment II-GPS*
  - *Experiment II-Modbus*
  - *Experiment II-Weather*
- **Analysis III**: Analysis III focuses instead on the implementation of SAGE in order to determine which features contribute the most to model performance. We provide observations from our SAGE plots for our best performing model

and compare these outputs with that of the best performing benchmark models. Our objective for this analysis is to explain the performance of our best models with respect to feature importance. In addition to providing SAGE plots to determine feature importance, we further solidify the importance of a certain features in our dataset through an ablation study. We provide tables from selected models in order to correlate feature importance with our visualizations with SAGE.

1.4.5 Analysis I

Analysis I provides the results of our proposed non-recurrent RIoT with the non-recurrent RL benchmarks on our telemetry datasets. In addition, we provide bar charts which detail the amount of correctly estimated, false negative, and false positive for our best performing models. We also provide confusion matrices which detail which classes of traffic were misclassified as other classes of traffic.

Experiment I-GPS

Table 1.2 details the results of applying the base models on the TON-IoT GPS dataset. Notably, the models combined with an adversarial environment agent tended to perform better than those base ML models and base RL models. The best model with respect to the G-mean was the AE-Dueling DQN, which also had the best performance on the F1-score. Granted, the RIoT and base AE-RL had the best performance on both the precision and recall metrics, respectively. Notably, the MLP had the best performance of accuracy, however; we have decided to not prioritize accuracy due to the imbalances present in the dataset. There were no statistically significant differences between the best performing model, the AE-Dueling DQN, and the next best model, the AE-RL, however. We provide a visualization of our results of our best performing model using a bar chart and also provide a comparison of what attacks were classified correctly versus incorrectly in the confusion matrix as

Table 1.2: Results from the base models using the TON-IoT GPS dataset. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds.

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	62.86 (15.67)	73.35 (4.85)	**75.86 (4.35)**	75.13 (0.88)	72.39 (3.56)	74.54 (3.27)	71.61 (1.95)
Precision	61.87 (9.49)	75.02 (5.25)	76.61 (4.93)	70.28 (2.61)	72.01 (2.68)	70.76 (3.78)	**76.40 (5.47)**
Recall	62.86 (15.67)	73.35 (4.85)	75.86 (4.35)	75.13 (0.88)	**77.00 (1.99)**	71.84 (4.41)	71.61 (1.95)
F1-Score	59.72 (13.73)	72.81 (4.38)	75.20 (4.08)	71.08 (1.70)	72.35 (3.56)	**74.54 (3.27)**	71.50 (2.46)
G-Mean	71.81 (10.61)	83.82 (3.01)	84.14 (2.63)	84.39 (0.54)	84.33 (2.11)	**85.32 (2.04)**	83.90 (1.08)
Training Time	154.4 (7.02)	0.0078 (0.0034)	27.5 (2.92)	1690.9 (71.78)	4024.2 (268.97)	3762.3 (21.76)	4049.1 (281.43)

shown in Figure 1.6 and Figure 1.7. There is a greater tendency for our best model to overclassify instances of traffic as backdoor traffic, while underclassifying instances of injection, ransomware, and password attacks. With respect to training times, we observe that the fastest out of all models was the NB classifier, followed by the MLP and SVM. Our fastest RL model was the DQN. We observe similar training times for the AE-RL our novel RIoT; the AE-Dueling DQN which had the best F1-score and G-mean also demonstrated the fastest training time out of our RL models with adversarial environment agents.

Experiment I-Modbus

Table 1.3 details the results of applying the base models on the TON-IoT Modbus dataset. Notably, this dataset had several models fare significantly poorly compared with the previous GPS tracker dataset. The best model with respect to the G-mean was the benchmark DQN, which also had the statistically best performance on the all other metrics. Our other reinforcement learning benchmarks did not perform as well compared with the benchmark DQN; the best non-benchmark performance was the novel AE-Dueling DQN. We provide a more robust visualization of our results for our best performing model using both a bar chart and provide a comparison of

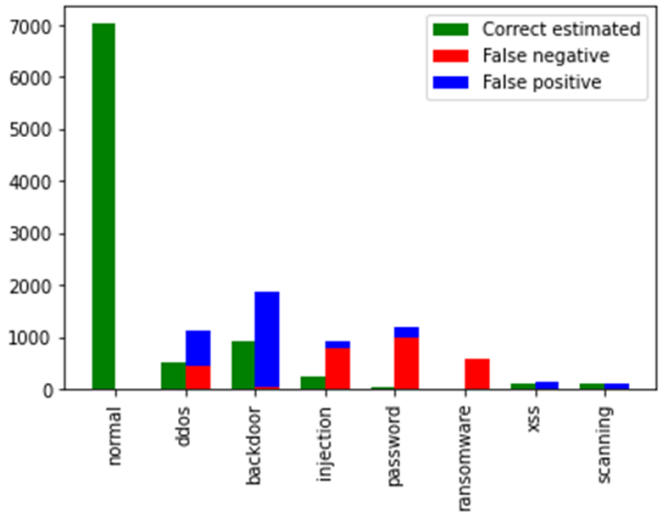


Figure 1.6: Bar chart of correctly identified, false positive, and false negative for the AE-Dueling DQN algorithm on the TON-IoT GPS dataset.

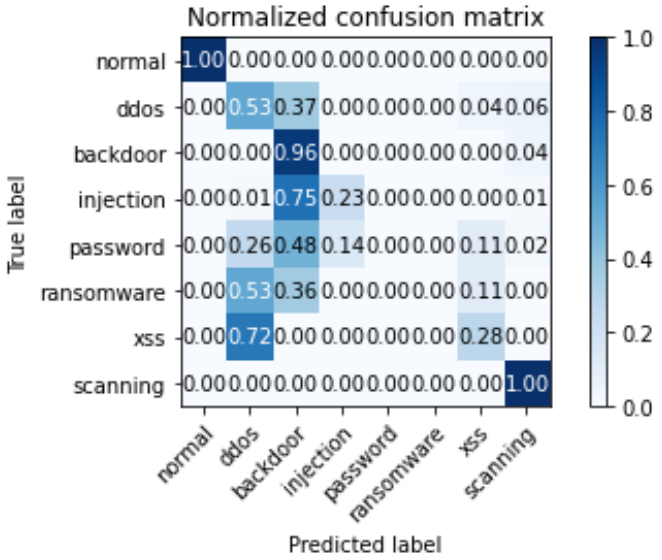


Figure 1.7: Confusion Matrix for the AE-Dueling DQN algorithm on the TON-IoT GPS dataset.

Table 1.3: Results from the base models using the TON-IoT Modbus dataset. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds.

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	68.72 (0.02)	68.98 (0.03)	42.39 (21.47)	**79.26* (0.36)**	14.86 (3.16)	73.27 (4.91)	27.38 (25.65)
Precision	47.22 (0.03)	47.59 (0.05)	51.75 (4.25)	**78.79* (0.36)**	49.98 (1.25)	77.00 (1.44)	30.65 (24.36)
Recall	68.72 (0.02)	68.98 (0.03)	42.39 (21.47)	**79.26* (0.36)**	14.86 (3.16)	73.27 (4.91)	27.38 (25.65)
F1-Score	55.98 (0.03)	56.32 (0.04)	38.99 (18.39)	**78.44* (0.55)**	19.37 (4.31)	74.19 (3.09)	22.23 (23.70)
G-Mean	46.36 (0.01)	46.26 (0.01)	45.52 (6.94)	**86.34 (0.34)**	35.35 (3.04)	85.12 (3.16)	36.31 (9.42)
Training Time	121.9 (3.83)	0.0719 (0.0048)	10.1 (1.50)	1868.3 (112.47)	4056.0 (271.17)	3772.7 (25.89)	3992.2 (192.49)

what attacks were classified correctly versus incorrectly in the confusion matrix as shown in Figure 1.8 and Figure 1.9. Our best model for this dataset instead tends to overclassify attacks as either backdoor, injection, or password attacks. Classification of network traffic as either XSS or scanning attacks was near non-existent. However, the best model was correctly classified all instances of normal traffic. Akin to the previous experiment with the TON-IoT GPS tracker, we observe the fastest time being the NB; the benchmark models had much faster training times as opposed to the RL and novel RL models, but generally tended to have worse performance. The DQN, which had the best performance across our performance metrics, was the fastest RL model.

Experiment I-Weather

Table 1.4 details performance across our models when applied to the TON-IoT Weather telemetry dataset. We observe the best classification performance utilizing this dataset. RIoT exhibited best performance compared with other ML models and RL algorithms on our metric of interest, the G-Mean, along wit the precision metric. The AE-RL exhibited the best performance across the other metrics of accuracy, recall, and F1-score. Akin to the GPS traffic is that we were not able to discern

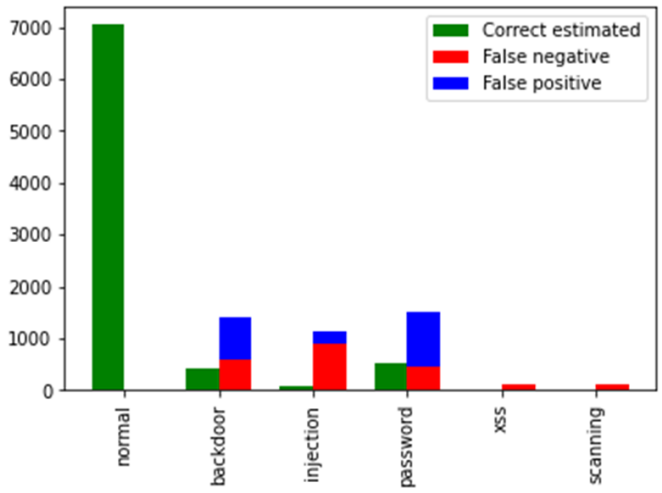


Figure 1.8: Bar chart of correctly identified, false positive, and false negative for the base DQN algorithm on the TON-IoT Modbus dataset.

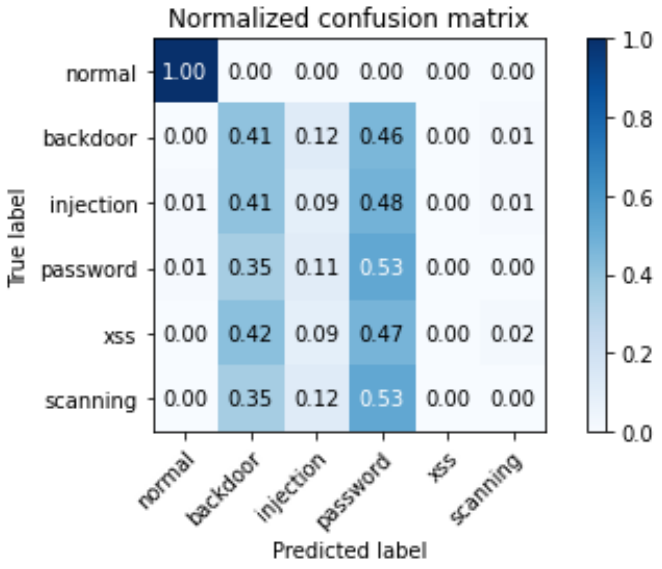


Figure 1.9: Confusion Matrix for the base DQN algorithm on the TON-IoT Modbus dataset.

Table 1.4: Results from the base models using the TON-IoT Weather dataset. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds.

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	75.66 (2.60)	74.02 (2.54)	77.96 (2.13)	72.81 (5.87)	**80.59 (1.31)**	73.19 (1.91)	80.23 (1.30)
Precision	77.11 (2.65)	75.95 (3.29)	78.51 (2.74)	68.84 (5.17)	82.89 (0.94)	68.23 (2.20)	**84.06 (2.11)**
Recall	75.66 (2.60)	74.02 (2.54)	77.96 (2.13)	72.81 (5.87)	**80.59 (1.31)**	73.19 (1.91)	80.23 (1.30)
F1-Score	75.73 (2.38)	73.28 (2.22)	77.44 (2.29)	69.86 (5.10)	**80.58 (1.28)**	69.27 (1.90)	80.17 (1.33)
G-Mean	85.06 (1.62)	84.11 (1.61)	86.46 (1.28)	84.26 (3.49)	88.72 (0.80)	84.58 (1.17)	**89.07 (0.85)**
Training Time	228.5 (7.59)	0.0753 (0.0014)	33.0 (2.37)	2081.9 (130.92)	4128.3 (360.29)	3781.9 (29.02)	3994.0 (302.19)

any statistically significant results from our observations. We provide a more robust visualization of our results using both the bar chart described in Figure 1.10 and provide a comparison of what attacks were classified correctly versus incorrectly in the confusion matrix as shown in Figure 1.11. We observe that RIoT performs exceptionally well on classification of normal traffic, while tending to generate more false positives on XSS and scanning traffics indicating the model classified more traffic as XSS and scanning compared with correct classification. RIoT struggled somewhat with correct classification of other traffic classes; there were several instances of DDoS traffic that was misclassified as another attack class, while several false-negatives were associated with password attacks. Again, similar results were observed with respect to training times of our models; that is, our benchmark models were faster as opposed to our RL algorithms. RIoT, which performed the best with respect to G-mean, was the second-to-slowest of all the models with respect to training time.

1.4.6 Analysis II

Analysis II provides the results of our recurrent RRIoT with the novel, recurrent RL algorithms with unique network architecture benchmarks on our IoT telemetry

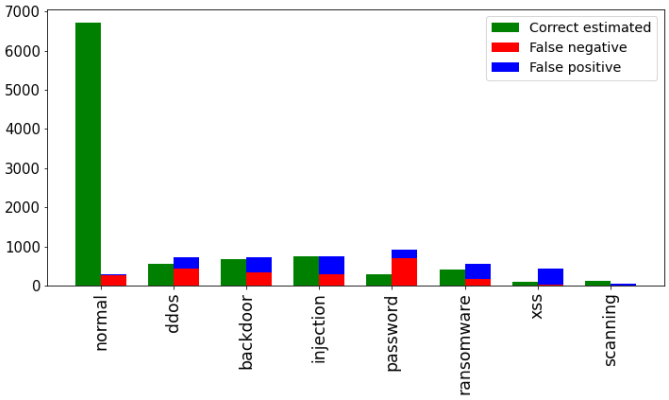


Figure 1.10: Bar chart of correctly identified, false positive, and false negative for RIoT on the TON-IoT weather dataset.

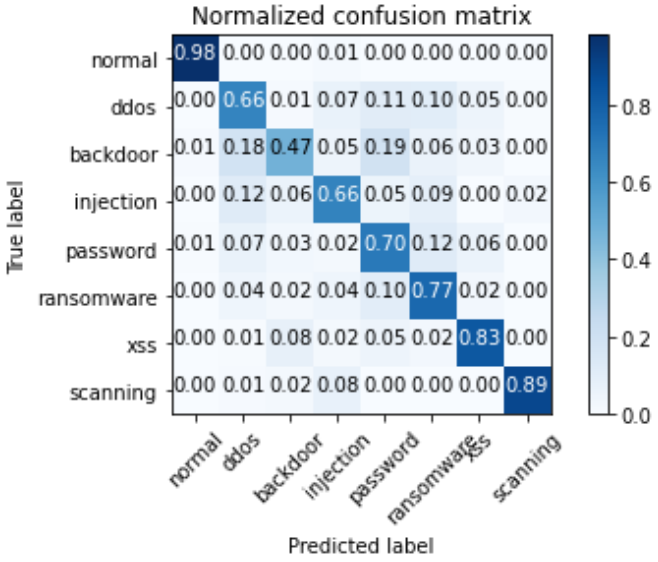


Figure 1.11: Confusion Matrix for RIoT on the TON-IoT weather dataset

datasets. We provide a graphical visualization of results using both bar charts and confusion matrices for our best performing RRIoT implementation for brevity.

Experiment II-GPS

Table 1.5 describes the results when applying recurrent layers to the our novel RL approach, RRIoT, and compares the performance against other novel RL benchmark models. Noteworthy in the performance is that again the AE-Dueling DRQN was the best model based upon the G-mean metric, akin to Experiment I-GPS wherein the AE-Dueling DQN was the best model. The performance of the AE-DRQN was statistically significant for the precision metric. Overall, the addition of an LSTM layer generally improved the performance across most of our performance metrics. On the AE-DRQN, we observe improved performance across all performance metrics as compared to the base AE-RL model. Additionally, for the AE-Dueling DRQN, we observe an improved accuracy, recall, F1-score, and G-mean when compared with the non-recurrent AE-Dueling DQN. When incorporating an recurrent layer to RIoT on both the actor and critic networks, we observe that the novel RRIoT-AC obtained a higher precision and F1-score compared to the base model. As observed, we find that our RRIoT-AC, which is the most complex model as it possesses recurrent layers on both the actor and critic on both the adversarial environment and classifier agent, has the longest training time as opposed to other training times on other models. Generally, we observe that as the complexity of the model increases, that is, as we include recurrent layers to the model, the longer the training time. We observe for the GPS model, that as we increase the complexity of the model, we tend to have better results; our AE-Dueling DRQN had the best G-mean score while not having the fastest training time for the RL models.

Table 1.5: Results from the base models compared with the recurrent models using the TON-IoT GPS dataset. The recurrent models are located on the leftmost part of the table. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds.

	Model							
	AE-RL	**AE-Dueling DQN**	**RIoT**	**AE-DRQN**	**AE-Dueling DRQN**	**RRIoT-A**	**RRIoT-C**	**RRIoT-AC**
Accuracy	72.39 (3.56)	74.54 (3.27)	71.61 (1.95)	73.82 (3.82)	**75.62 (2.91)**	68.36 (3.18)	71.31 (1.94)	70.29 (4.33)
Precision	77.00 (1.99)	71.84 (4.41)	76.40 (5.47)	**79.92* (1.76)**	75.36 (5.08)	76.27 (2.23)	77.69 (1.52)	78.63 (2.44)
Recall	72.39 (3.56)	74.54 (3.27)	71.61 (1.95)	73.82 (3.82)	**75.62 (2.91)**	68.36 (3.18)	71.31 (1.94)	70.29 (4.33)
F1-Score	72.01 (2.68)	70.76 (3.78)	71.50 (2.46)	**75.34 (3.07)**	73.29 (3.07)	70.24 (2.67)	71.34 (1.84)	72.63 (4.00)
G-Mean	84.33 (2.11)	85.32 (2.04)	83.90 (1.08)	84.88 (2.21)	**86.05 (1.71)**	81.34 (2.42)	84.04 (0.97)	83.59 (1.83)
Training Time	4024.2 (268.97)	3762.3 (21.76)	4049.1 (281.43)	5114.3 (502.91)	4002.0 (199.83)	4286.2 (68.56)	3864.6 (75.82)	5477.0 (212.62)

Experiment II-Modbus

Table 1.6 describes the results when applying recurrent layers to our novel RL model, RRIoT, and compares the performance against other novel RL benchmark models on the Modbus telemetry dataset. Noteworthy compared with the previous experiment utilizing an LSTM layer is the absence of universal improvements in all models when adding a recurrent layer. There were no statistically significant results when comparing the non-recurrent model versus the recurrent layers. There were some limited improvements across our models. We see that adding recurrent layers only improved some metrics on some models, namely the accuracy, recall, and F1-score for the improvement from an AE-Dueling DQN to AE-Dueling DRQN and on all performance metrics to include precision and G-mean when moving from the non-recurrent RIoT to RRIoT with an critic recurrent network. The other recurrent RL models to include the AE-DRQN and the RRIoT with both actor and actor-critic recurrent networks performed either as well or worse than that of their non-recurrent counterparts. As observed similarly in the previous experiment, we find that the fastest RL model is our AE-Dueling DQN; however, the fastest recurrent RL model is our RRIoT with a recurrent critic network. We observe that our RRIoT-AC has the longest training time. The AE-Dueling DRQN, which was the best performing recurrent model, had training times similar to that of the other novel RL benchmarks.

Table 1.6: Results from the base models compared with the recurrent models using the TON-IoT Modbus dataset. The recurrent models are located on the leftmost part of the table. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds.

	Model							
	AE-RL	**AE-Dueling DQN**	**RIoT**	**AE-DRQN**	**AE-Dueling DRQN**	**RRIoT-A**	**RRIoT-C**	**RRIoT-AC**
Accuracy	14.86 (3.16)	73.27 (4.91)	27.38 (25.65)	14.38 (14.13)	**74.54 (2.50)**	24.39 (29.35)	55.64 (26.43)	10.33 (20.89)
Precision	49.98 (1.25)	**77.00 (1.44)**	30.65 (24.36)	32.15 (24.34)	76.56 (1.11)	29.91 (25.12)	45.65 (17.43)	34.55 (31.05)
Recall	14.86 (3.16)	73.27 (4.91)	27.38 (25.65)	14.38 (14.13)	**74.54 (2.50)**	24.39 (29.35)	55.64 (26.43)	10.33 (20.89)
F1-Score	19.37 (4.31)	74.19 (3.09)	22.23 (23.70)	12.36 (16.96)	**74.25 (1.38)**	18.06 (25.44)	44.71 (23.04)	34.55 (31.05)
G-Mean	35.35 (3.04)	**85.12 (3.16)**	36.31 (9.42)	30.33 (12.24)	83.69 (1.99)	29.91 (13.76)	41.31 (12.20)	20.02 (12.13)
Training Time	4128.3 (360.29)	3781.9 (29.02)	3994.0 (302.19)	4548.5 (803.68)	4106.3 (21.38)	3780.8 (24.18)	3870.7 (79.99)	5861.1 (274.60)

Overall, we find our RL models have training times more closer to each other; there is less distinction in training times across our RL models.

Experiment II-Weather

In this section, we describe the results when applying recurrent layers to our novel RL model, RRIoT, and compares the performance against other novel RL benchmark models on the weather telemetry dataset. The best performance was observed in RRIoT with both recurrent actor and critic networks. RRIoT exhibited the statistically best performance on the precision, F1-score, and G-mean metrics. Overall, the addition of an LSTM layer improved the performance across most of our models. We additionally observe a tighter confidence interval for these recurrent models. RRIoT demonstrates a solid performance in classification of all forms of traffic, both normal and attack traffic. False positives and false negatives were kept relatively to a minimum when compared to previous experiments. For our training time analysis for the weather dataset, we find that while our RRIoT with actor and critic recurrent layers had the longest training time, it obtained the best performance on our metrics. Training times for our novel recurrent RL models such as RRIoT were similar to that of our novel RL benchmarks for the weather telemetry data. We provide an overview of our results in Table 1.7, and we provide a more robust

Table 1.7: Results from the base models compared with the recurrent models using the TON-IoT Weather dataset. The recurrent models are located on the leftmost part of the table. Statistically significant results are indicated by an asterisk (*). Training times are reported in seconds.

	Model							
	AE-RL	**AE-Dueling DQN**	**RIoT**	**AE-DRQN**	**AE-Dueling DRQN**	**RRIoT-A**	**RRIoT-C**	**RRIoT-AC**
Accuracy	80.59 (1.31)	73.19 (1.91)	80.23 (1.30)	79.46 (1.94)	73.91 (2.51)	84.63 (1.36)	80.01 (2.00)	**85.22 (1.10)**
Precision	82.89 (0.94)	68.23 (2.20)	84.06 (2.11)	82.97 (1.72)	71.61 (5.57)	86.07 (0.87)	83.63 (1.66)	**86.55* (0.80)**
Recall	80.59 (1.31)	73.19 (1.91)	80.23 (1.30)	79.46 (1.94)	73.91 (2.51)	84.63 (1.36)	80.01 (2.00)	**85.22 (1.10)**
F1-Score	80.58 (1.28)	69.27 (1.90)	80.17 (1.33)	79.55 (1.92)	70.64 (3.78)	84.83 (1.20)	79.99 (2.16)	**85.44* (1.03)**
G-Mean	88.72 (0.80)	84.58 (1.17)	89.07 (0.85)	88.36 (1.18)	84.92 (1.82)	90.86 (0.85)	88.66 (1.03)	**91.41* (0.56)**
Training Time	4056.0 (271.17)	3772.66 (25.89)	3992.2 (192.49)	4981.5 (507.34)	4090.3 (14.15)	3780.4 (22.78)	3850.4 (75.55)	5700.0 (394.95)

visualization of our results using both the bar chart described in Figure 1.12 and provide a comparison of what attacks were classified correctly versus incorrectly in the confusion matrix as shown in Figure 1.13.

1.4.7 Analysis III

In the following section, we present the SAGE plots for our best and second best RL models per experiment based upon G-mean performance. As mentioned previously, SAGE provides a global explanation as to the amount of contribution a feature to a model's performance. For this analysis, we mainly consider the magnitude of importance for each feature on model performance.

We provide an overview of the SAGE plots for each of our non-recurrent RL models, the AE-RL, AE-Dueling DQN, and RIoT, when implemented on the GPS, Modbus, and weather telemetry datasets, respectively. When comparing magnitude across the models we observe that generally, the AE-Dueling DQN has the most granular magnitudes, while the AE-RL model tends to have the largest magnitudes. These graphical results indicate that the feature importance on the AE-Dueling DQN is more granular in feature importance with respect to contribution to model performance than other models, while the features for the AE-RL model tend to contribute greater with respect to model performance.

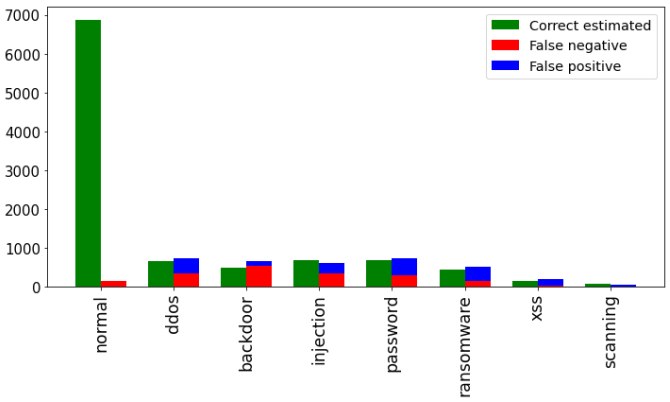


Figure 1.12: Bar chart of correctly identified, false positive, and false negative for RRIoT on the TON-IoT weather dataset.

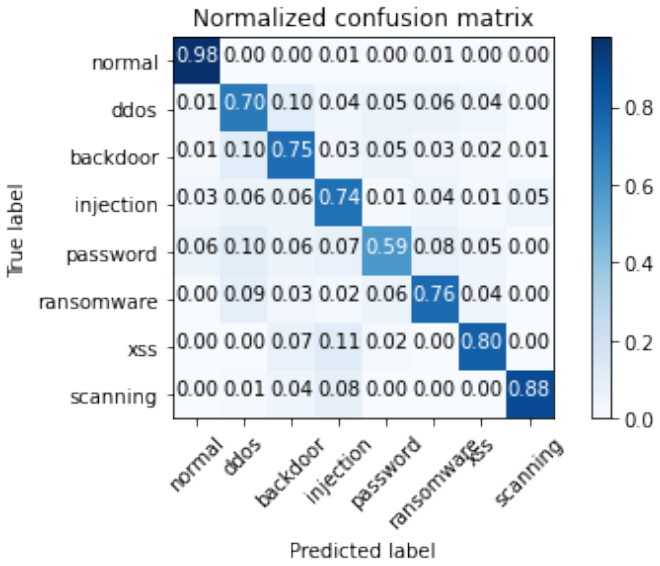


Figure 1.13: Confusion Matrix for RRIoT on the TON-IoT weather dataset.

In addition, we also provide an overview of the SAGE plots for each of our RL models with an incorporated recurrent layer. These figures represent the AE-DRQN, AE-Dueling DRQN, and RRIoT models with different novel recurrent architectures, to include the RRIoT-A, RRIoT-C, and RRIoT-AC when implemented on the GPS, Modbus, and weather telemetry datasets, respectively. When comparing magnitude across the models we observe that compared with the non-recurrent models, there tends to be less variability in terms of magnitude. We do, however, see a greater range with respect to the confidence intervals in the importance of each feature towards model performance. We discuss more concerning the SAGE output in section 1.5.

In order to better analyze the effect of features on model performance, we also provide an ablation study that links our SAGE plots with performances of our selected models. Our numerical tables are presented in section 1.5.2, along with providing an experiment on intermittent data in section 1.5.3. Noticeably, on our GPS dataset, we observe a notable decrease in model performance when the latitude feature was withheld. This performance thus correlates to our SAGE plots; the latitude feature generally had the greatest magnitude out of any SAGE plot. Our Modbus dataset with the first register withheld had a negative impact on our models' performances; however, there were other features with greater magnitude on our SAGE plots other than the first register. Additionally, for the Weather telemetry dataset, when the humidity feature was withheld, a heavily discernible decrease in performance was observed across the models; the humidity feature had the greatest magnitude observed in our SAGE plots. We observed a decrease across all models with a reduced feature set which correlates to the magnitudes of the features in our SAGE plots. Similar to withholding features for an ablation study, we demonstrate that intermittent telemetry data decreases the performances of both our benchmark ML and novel RL models.

1.5 Discussion

From our performance results across our benchmark RL and novel benchmark RL models when compared with our novel RIoT, we find that RIoT tends to perform well on the weather telemetry dataset. Correspondingly, our novel benchmarks, the AE-Dueling DQN and AE-Dueling DRQN was the best model on both the GPS and Modbus telemetry datasets. We do observe that RIoT was the second best model on the Modbus telemetry dataset. RRIoT exhibited superior performance on the weather dataset; the actor recurrent and actor and critic recurrent networks on RRIoT were the second and first best models when analyzing that dataset, respectively. Interestingly, however, is that even though RIoT and RRIoT did not perform as well as the other novel benchmark RL algorithms, RIoT and RRIoT had "cleaner" confusion matrices in that there were less prominent instances of traffic being misclassified as other traffic (for example, in the Experiment-I-GPS, we see that the AE-Dueling DQN tended to misclassify traffic as backdoor, whereas in Experiment-I-Weather, we observe a high proportion of observations falling on the diagonal for our confusion matrix). This observation indicates that RIoT and RRIoT are able to distinguish the different classes and make accurate predictions.

With respect to the training times of our models we observe that our fastest models were our benchmark models. Our novel RL models needed more time in order to train as opposed to our benchmarks. While our benchmark models were the fastest, our experiments across both analysis I and II demonstrate that they tended to have significantly worse results across our performance metrics when compared to our novel RL models. The fastest model to train, the NB classifier, usually presented with some of the worst performance on classification. As a result, while we acknowledge the faster training times of our benchmarks, we demonstrate that having a longer time as demonstrated tends to increase our model performance.

We better strengthen our discussion section by including additional analyses. We determine feature importance based upon performance through the implementation

of SAGE. Furthermore, we also include a subset of experiments for an ablation study in the following sections in order to further demonstrate the importance of each feature for our IoT devices. We demonstrate our models' performances on intermittent data, in which we have a full feature space but a subset of observations are instead rendered incomplete. We end our discussion with including other considerations for our algorithms, especially with respect to direct deployment on IoT devices. Additional considerations, such as test dataset sensitivity and SAGE plots, are in Appendix A.1 and Appendix A.2, respectively.

1.5.1 SAGE Analysis

Figures 1.14 and 1.15 provide a comparison of the AE-Dueling DQN model presented in Figure 1.14, which was the best performing model, to the AE-RL model's performance on the GPS telemetry dataset presented in Figure 1.15. Observed is the AE-Dueling DQN's ability to be more granular compared to the AE-RL model in terms of which features were the most important for performance. Compared with the AE-RL model, we also observe that the latitude variable was more important in considering model performance for the AE-Dueling DQN, whereas the AE-RL model had both models roughly equivalent for contribution to model performance. The AE-RL model also had somewhat larger variability compared to the AE-Dueling DQN model. These differences in the SAGE plots suggest that the differences in feature importance and magnitude contributed to the differences in the model performances.

Figures 1.16 and 1.17 provide a comparison of the AE-Dueling DQN model presented in Figure 1.16, which was the best performing model again, to RIoT's performance on the Modbus telemetry dataset presented in Figure 1.17. Observed again is the AE-Dueling DQN's ability to be more granular compared to our novel RIoT for feature importance; similar performance was observed for RIoT on the GPS telemetry dataset. Notably, there are more features for our Modbus telemetry dataset which include four Modbus registers which themselves contain data. Compared with

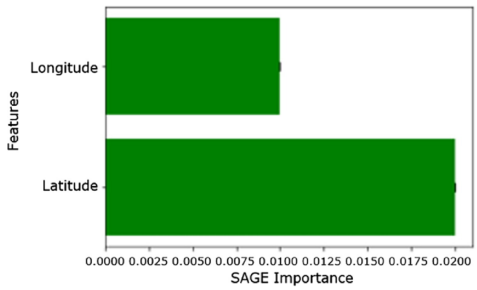


Figure 1.14: SAGE Plot for the AE-Dueling DQN Algorithm for Analysis I using the GPS telemetry dataset.

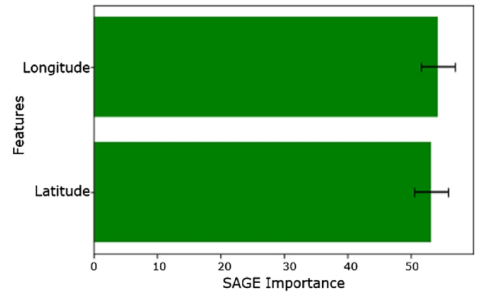


Figure 1.15: SAGE Plot for the AE-RL Algorithm for Analysis I using the GPS telemetry dataset.

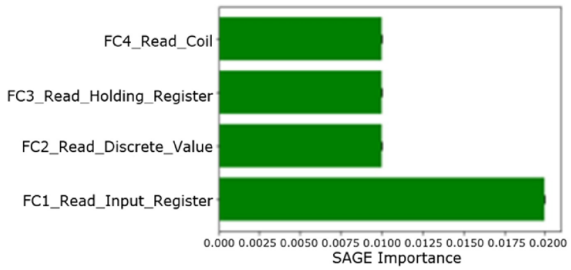


Figure 1.16: SAGE Plot for the AE-Dueling DQN Algorithm for Analysis I using the Modbus telemetry dataset.

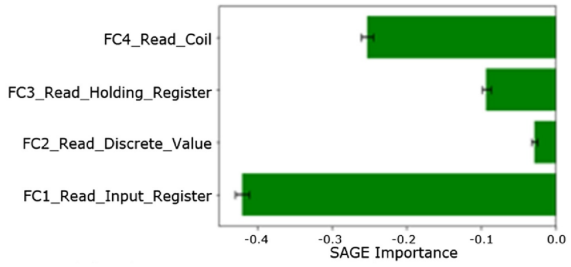


Figure 1.17: SAGE Plot for RIoT for Analysis I using the Modbus telemetry dataset.

RIoT, we observe that the AE-Dueling DQN weights three out of the four registers with equal weight, while RIoT had more variability in determining feature importance. RIoT also had larger confidence intervals compared to the AE-Dueling DQN model. We consider that the equal importance of the three out of the four Modbus registers contributed to a better performance for the AE-Dueling DQN.

Figures 1.18 and 1.19 provide a comparison of our novel RIoT presented in figure 1.18, which was the best performing model again, to the AE-RL model's performance on the Modbus telemetry dataset presented in Figure 1.19. As compared to the previous Modbus SAGE plots, we observe that the variation in feature importance as reflected by RIoT is indicative of better model performance. RIoT and AE-Dueling DQN both had weighted humidity as the most importance, however, there is variation in the feature importance considering the temperature and humidity. While the AE-Dueling DQN considers both roughly equal, RIoT has the pressure feature with greater magnitude for contribution towards model performance.

Figures 1.20 and 1.21 provide a comparison of the AE-Dueling DRQN model presented in Figure 1.20, which again was the best performing model, to the AE-DRQN model's performance on the GPS telemetry dataset presented in Figure 1.21. Akin to the analysis I on the GPS telemetry dataset, we see that our AE-Dueling DRQN model performs best. Unlike the previous non-recurrent analysis, we find that both the AE-Dueling DRQN and AE-DRQN models had the dataset features both roughly equal in granularity of magnitude of contribution towards model performance. The AE-Dueling DRQN had both features roughly equal in magnitude; however, when using the AE-DRQN, the longitude feature contributed more to the model performance. We also observe a confidence interval for the latitude feature for the AE-DRQN.

Figures 1.22 and 1.23 provide a comparison of the AE-Dueling DQN model presented in Figure 1.22, which was the best performing model again, to RRIoT's performance on the Modbus telemetry dataset presented in Figure 1.23. We observe a similar granularity across both the AE-Dueling DRQN and RRIoT with respect

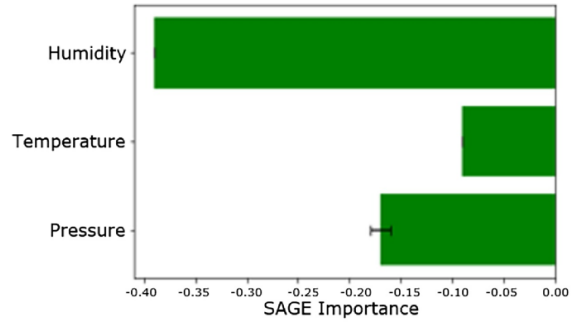


Figure 1.18: SAGE Plot for RIoT for Analysis I using the weather telemetry dataset.

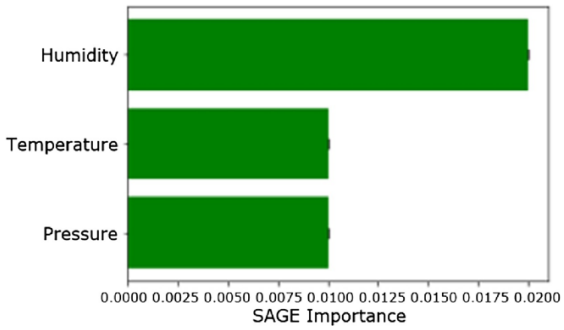


Figure 1.19: SAGE Plot for the AE-Dueling DQN Algorithm for Analysis I using the weather telemetry dataset.

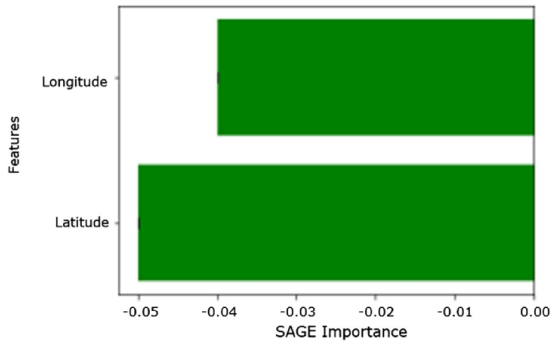


Figure 1.20: SAGE Plot for the AE-Dueling DRQN Algorithm for Analysis II using the GPS telemetry dataset.

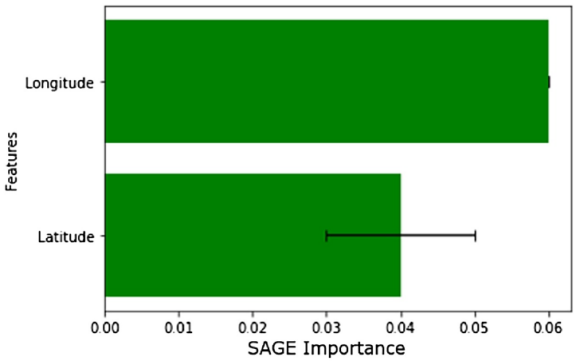


Figure 1.21: SAGE Plot for the AE-DRQN Algorithm for Analysis II using the GPS telemetry dataset.

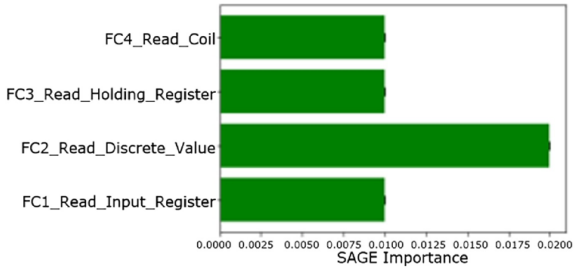


Figure 1.22: SAGE Plot for the AE-Dueling DRQN Algorithm for Analysis II using the Modbus telemetry dataset.

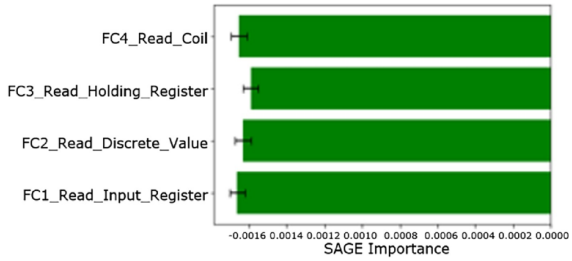


Figure 1.23: SAGE Plot for RRIoT for Analysis II using the Modbus telemetry dataset.

to the feature importance for model performance. In addition, we observe that the AE-Dueling DRQN rates the third register as the most important towards model performance, whereas RRIoT tends to rate all the features as roughly equally important towards model performance. RRIoT also has wider confidence intervals compared to the AE-Dueling DRQN.

Figures 1.24 and 1.25 provide an overall comparison of our novel RRIoT-A presented in Figure 1.24, which was the best performing model again, to RRIoT-AC's performance on the weather telemetry dataset presented in Figure 1.25. Notably, in this experiment, we observe that our novel RRIoT was superior in comparison with the other novel benchmark RL algorithms. RRIoT with both actor and critic recurrent networks outperformed all other models; the next best model was RRIoT with actor recurrent. Consequentially, we observe the SAGE plots for both models being roughly similar in nature. Granularity and spread of confidence intervals were roughly similar for both models. The difference between the two models in terms of which feature contributes the most to model performance is that RRIoT-AC had a greater magnitude for all of its features compared to RRIoT-A.

1.5.2 Ablation Study

In addition to our SAGE analysis described above, we additionally include a subset of experiments for an ablation study. Specifically, for these ablation experiments, we opt to withhold features in order to determine model performance with an incomplete feature space. Specifically, we withhold a single feature from an already reduced feature for analysis. We provide the results from our reduced feature set in the following tables.

Table 1.8 describes the results when withholding a single feature from our GPS dataset, namely, the latitude feature, before feeding the data into the models. Noteworthy in the performance is that again the AE-Dueling DQN was the best

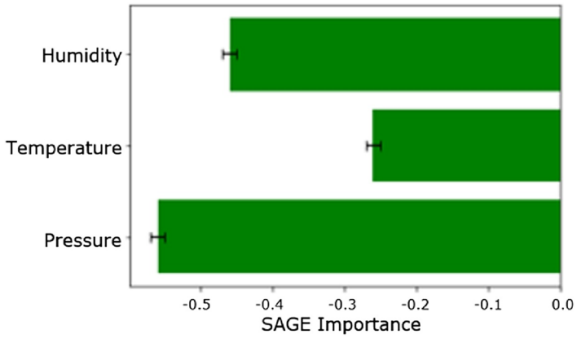


Figure 1.24: SAGE Plot for RRIoT-AC for Analysis II using the Weather telemetry dataset.

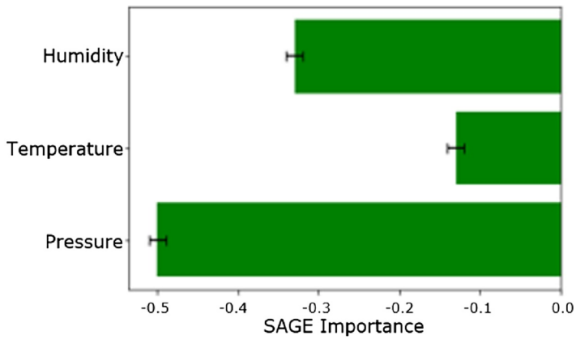


Figure 1.25: SAGE Plot for RRIoT-A for Analysis II using the Weather telemetry dataset.

Table 1.8: Results from the base models using the TON-IoT GPS dataset wherein a single feature, in this instance, latitude, withheld from the model. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	52.89 (36.86)	64.26 (36.62)	62.07 (39.43)	65.25 (1.24)	68.74 (3.57)	**70.58 (4.32)**	67.92 (1.46)
Precision	48.52 (37.73)	64.75 (35.94)	60.36 (40.48)	55.58 (5.09)	66.67 (9.97)	**75.94 (1.70)**	65.04 (1.46)
Recall	52.89 (36.86)	64.26 (36.62)	62.07 (39.43)	65.25 (1.24)	68.74 (3.57)	**70.58 (4.32)**	67.92 (1.46)
F1-Score	49.84 (37.57)	64.22 (36.86)	60.74 (39.43)	56.34 (3.76)	64.64 (5.95)	**70.03 (2.93)**	64.84 (1.19)
G-Mean	31.25 (25.86)	34.80 (34.63)	30.48 (35.39)	68.69 (5.89)	80.37 (7.64)	**82.87 (2.64)**	81.68 (0.91)

model based upon the G-mean metric, akin to Experiment I-GPS wherein the AE-Dueling DQN was the best model. Our AE-Dueling DQN demonstrated the best performance across all metrics, the second best model was our novel RIoT followed by AE-RL in terms of G-mean. Of special note is that while our model performances decreased significantly compared to their baselines, our novel RL models exhibited a tighter spread in results.

Table 1.9 describes the results when withholding a single feature from our Modbus dataset, namely, the first Modbus register, before feeding the data into the models. Akin to the Analysis I and Analysis II, the base DQN exhibited the best performance on the G-mean, accuracy, and recall metrics. Interestingly, our NB classifier obtained the highest precision and F1-scores, and had the second best G-mean performance after our base DQN. Compared to the previous ablation experiment with the GPS dataset, removal of a single feature for the TON-IoT Modbus telemetry dataset had a significantly detrimental effect on our novel RL model performances.

We also provide results when withholding a single feature from our weather dataset, namely, the humidity feature, before feeding the data into the models. Similar to our previous analyses with the TON-IoT weather dataset, our RIoT exhibited the best performance with respect to our preferred metric, G-mean, and accuracy and recall. Noteworthy is that our AE-Dueling DQN had the best performance on the

Table 1.9: Results from the base models using the TON-IoT Modbus dataset wherein a single feature, in this instance, the first input register, withheld from the model. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	68.49 (40.19)	68.49 (0.08)	51.75 (34.70)	**77.80 (1.15)**	24.44 (9.45)	13.12 (5.04)	27.84 (28.26)
Precision	90.00 (30.00)	**99.99 (00.00)**	60.41 (38.29)	69.91 (3.55)	49.39 (2.23)	51.33 (10.84)	31.70 (2.66)
Recall	68.49 (40.19)	68.49 (0.08)	51.75 (34.70)	**77.80 (1.15)**	24.44 (9.45)	13.12 (5.04)	27.84 (28.26)
F1-Score	72.18 (38.49)	**81.29 (0.58)**	49.77 (36.88)	72.15 (2.05)	28.03 (13.27)	10.74 (9.79)	18.84 (25.72)
G-Mean	79.25 (36.10)	84.25 (0.41)	43.00 (26.64)	**84.59 (4.99)**	41.55 (6.76)	32.98 (5.23)	35.18 (7.76)

precision and F1-score metrics. Compared with the previous ablation experiment with the GPS dataset, our novel RL models demonstrated the best results compared to our benchmark ML models. Additionally, our novel RL models, save for the AE-RL model, tended to exhibit a tighter spread in results as opposed to the performance observed in benchmark ML models. Table 1.10 describes our observed results.

1.5.3 Intermittent Traffic

In addition to applying an ablation study in order to examine the implications of having an incomplete feature space, we additionally provide an analysis of intermittent traffic. We consider the scenario in which approximately 20% of observations across the feature space posses incomplete data. An example of an intermittent feature set is depicted in Table 1.11. We provide the results of applying intermittent traffic in the following tables.

Table 1.12 details the results of applying our models on the TON-IoT GPS dataset wherein approximately 20% of observations are incomplete. Our results indicate that one of our benchmark ML models, the MLP, performs the best across all performance metrics, and performs statistically better in G-mean, accuracy, and recall. We observe that our benchmark ML models performed better compared to our novel RL models. While our RIoT did not have the best performance in comparison to other novel

Table 1.10: Results from the base models using the TON-IoT Weather dataset wherein a single feature, in this instance, humidity, withheld from the model. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	60.76 (37.59)	60.92 (37.98)	62.89 (35.91)	70.17 (1.35)	62.62 (27.13)	75.28 (9.42)	**75.37 (2.66)**
Precision	60.97 (37.10)	64.00 (39.21)	62.77 (37.78)	61.28 (3.10)	66.96 (30.20)	**80.16 (5.03)**	77.92 (2.85)
Recall	60.76 (37.59)	60.92 (37.98)	62.89 (35.91)	70.17 (1.35)	62.62 (27.13)	75.28 (9.42)	**75.37 (2.66)**
F1-Score	60.49 (37.50)	60.96 (38.82)	61.90 (37.05)	64.08 (2.03)	62.13 (28.09)	**76.54 (7.89)**	74.59 (2.55)
G-Mean	33.74 (30.95)	30.35 (33.99)	32.44 (34.32)	80.30 (3.45)	73.51 (27.35)	85.35 (6.56)	**86.02 (1.45)**

Table 1.11: A sample of observations of the intermittent dataset for the GPS telemetry data

Time Stamp	**Longitude**	**Latitude**
00015	116.5217	132.1625
00018	0	135.0725
00021	121.7959	135.1102
00024	116.6112	132.2246
0	121.8415	135.1514
00030	116.6469	0

Table 1.12: Results from the base models using the TON-IoT GPS dataset wherein specific observations are incomplete. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	72.08 (0.32)	76.03 (0.65)	**79.94*** **(2.13)**	78.27 (0.47)	70.95 (4.71)	71.22 (0.42)	64.51 (1.85)
Precision	81.00 (1.17)	79.93 (0.74)	**81.81** **(2.01)**	71.72 (1.70)	76.41 (2.93)	73.69 (0.53)	75.02 (1.73)
Recall	72.08 (0.32)	76.03 (0.65)	**79.94*** **(2.13)**	78.27 (0.47)	70.95 (4.71)	79.28 (1.08)	64.51 (1.85)
F1-Score	75.53 (0.51)	76.02 (0.70)	**80.49*** **(2.07)**	73.22 (0.43)	70.68 (3.17)	71.22 (0.42)	67.14 (1.47)
G-Mean	82.11 (0.51)	86.42 (0.41)	**88.72*** **(1.25)**	86.49 (2.49)	83.11 (3.23)	83.44 (0.29)	78.83 (1.22)

RL models, the standard deviation of our novel RIoT was smaller than the best performing model, the MLP.

Table 1.13 details the results of applying our models on the TON-IoT Modbus dataset wherein approximately 20% of observations are incomplete. Akin the previous experiment with the GPS telemetry dataset, we find that our benchmark models tended to perform better than our novel RL models. Specifically, we find that our benchmark SVM and NB classifier had the statistically best performance on the precision, F1-score, and G-mean metrics, while our base DQN had the statistically best performance on the accuracy and recall metrics.

Table 1.14 details the results of applying our models on the TON-IoT Weather dataset wherein approximately 20% of observations are incomplete. We observe that the benchmark ML models again perform the best compared to our novel RL models. Specifically, we find that the MLP classifier outperformed the other models on the precision, F1-score, and the G-mean metrics.

1.5.4 Considerations for Direct Deployment to IoT and Embedded Devices

While we consider our models to be executed primarily on a centralized server or workstation separate from the edge IoT devices, there is an interest in deploying these models to resource-constrained systems. As our models are deep RL models, we acknowledge that they require significant amount of computational power in order for execution. Our models run comfortably on 32GB of RAM with an i7 processor, but this hardware is not always present for IoT devices. It would be challenging to fit our ML benchmarks and novel RL models directly on embedded IoT systems. Resource intense algorithms are challenging for FPGAs and embedded devices for ML model deployment [3]. As described in [15], issues such as execution time, memory, power consumption, model accuracy, health, and scalability issues are challenges for model deployment on embedded systems.

Table 1.13: Results from the base models using the TON-IoT Modbus dataset wherein specific observations are incomplete. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	68.49 (0.29)	68.49 (0.56)	61.16 (11.80)	**77.18* (2.83)**	18.00 (2.55)	9.92 (9.47)	20.34 (23.71)
Precision	**99.99* (0.00)**	**99.99* (0.00)**	80.20 (20.13)	68.14 (7.20)	14.07 (20.34)	7.91 (13.19)	49.76 (7.81)
Recall	68.49 (0.29)	68.49 (0.56)	61.16 (11.80)	**77.18* (2.83)**	18.00 (2.55)	10.10 (18.34)	20.34 (23.71)
F1-Score	**81.29* (0.20)**	**81.29* (0.39)**	67.56 (17.22)	71.01 (5.07)	11.95 (21.91)	9.92 (9.47)	15.37 (20.57)
G-Mean	**84.25* (0.15)**	**84.25* (0.28)**	69.21 (12.98)	81.88 (12.15)	25.29 (13.68)	25.90 (11.50)	30.43 (11.60)

Table 1.14: Results from the base models using the TON-IoT Weather dataset wherein specific observations are incomplete. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	**79.08 (0.24)**	74.18 (0.63)	79.05 (1.30)	76.53 (3.04)	50.22 (2.34)	61.50 (3.29)	53.32 (3.97)
Precision	80.12 (0.29)	79.14 (0.63)	**81.13 (1.66)**	67.20 (6.80)	65.74 (1.83)	75.10 (1.43)	67.33 (0.47)
Recall	**79.08 (0.24)**	74.18 (0.63)	79.05 (1.30)	76.53 (3.04)	50.22 (2.34)	61.50 (3.29)	53.32 (3.97)
F1-Score	79.44 (0.24)	75.05 (0.63)	**79.54 (1.27)**	70.11 (5.20)	54.91 (2.22)	64.87 (3.07)	57.95 (2.91)
G-Mean	88.16 (0.15)	85.36 (0.40)	**88.21 (0.78)**	79.17 (12.95)	67.76 (1.77)	76.72 (2.11)	70.00 (2.32)

Ultimately, along with the feature analysis provided by SAGE, we observe that each telemetry dataset by each IoT device analyzed requires its own RL model to best detect attacks within its data. That is, the GPS telemetry data would be best analyzed by the AE-Dueling DQN, the Modbus telemetry data would be best analyzed by AE-Dueling DQN and the weather telemetry data would be best analyzed by RIoT. This behaviour was also demonstrated in our ablation studies in which features were withheld from our models to demonstrate the performance with an incomplete feature space. Potentially, this observation could be a result of the varying nature of the telemetry data, as while each of the telemetry datasets had a roughly similar distribution across traffic classes, not all telemetry data contained the same number or features.

With respect to model training times, we find that our benchmark ML models demonstrated the fastest training times. While the benchmark models were the fastest, they did not obtain the best performance; our novel RL models statistically outperformed our benchmarks. There are instances in which another model would work in the event that the preferred method is not present. We observe that RIoT performed second best compared with the other RL models for the Modbus telemetry dataset, and the AE-RL was the second best models on the GPS and weather datasets, while the AE-DRQN was the second best on the GPS telemetry dataset when compared to the other recurrent RL models. In these instances, it is best to utilize our novel RL models as opposed to the presented benchmark models.

We do observe that the performance of our novel RL models tended to decrease when IoT device telemetry data is intermittent or incomplete. This performance does indicate that our benchmark models perform better in instances of incomplete data. However, as stated previously, the performance of our novel RL models was better compared to benchmark ML models with an incomplete feature space. Overall, we do observe that our RL models generally tended to perform better than their ML benchmarks. Our RL models thus are a promising approach in order to provide better defenses for intrusion detection, especially on IoT systems.

1.6 Conclusion and Future Works

Proper cybersecurity depends on robust defenses against several attacks. Cyber attacks attempt to subvert the three pillars of information security, including confidentiality, integrity, and availability. The proliferation of embedded systems and IoT devices has also subjected these devices to cyber attacks, prompting a need for the detection of cyber attacks on these smaller devices. This study presents several methods to detect cyber attacks on IoT systems through the usage of RL algorithms.

In particular, this study presents RL models that are improved through the addition of recurrent layers in the network structures when combined with an adversarial environment as initially proposed in [16]. We improve upon popular Q-learning RL algorithms, to include the DQN and dueling DQN, and improve upon the DDPG algorithm to subsequently craft the AE-DRQN, AE-Dueling-DRQN, and RRIoT models. Our results show that our recurrent models generally had better performance compared with base ML and RL models when analyzing output from IoT devices. From our observations, the dueling DQN was superior in GPS traffic, the base DQN was superior in Modbus traffic, and the DDPG algorithm was best at determining attacks with weather traffic. These observations indicate that while inclusion of recurrent networks in our RL algorithms generally improves performance, specific RL models much be selected dependent upon IoT device for the best performance in detection of cyber attacks. Additionally, we also provide SAGE plots to better understand the decision making process for our models and provide insight into which features contribute to our models' performance which is reinforced by our ablation study.

With respect to future research, there are several potential directions. First, we can apply these RL algorithms towards other problems in the cybersecurity domain, especially in more nuanced fields such as deep packet inspection and malware detection. These RL algorithms could then simulate a more robust network-intrusion detection system or antivirus software. A second promising approach would be to

consider encrypted traffic. Cyber attacks often occur through the usage of encrypted protocols, thus presenting a need for an intrusion detection system that can properly detect threats with encrypted communications. Deployment of the RL algorithms on individual IoT devices would also be a potentially promising future approach; this would then serve as a potential host-based intrusion detection system as opposed to having the traffic aggregated and analyzed on a centralized server. As we have now demonstrated that recurrent models outperform non-recurrent models in our work, we can also consider the optimal amount of historical data needed for effective detection. Additionally, while in this work we applied an adversarial environment consisting of an intelligent environmental agent that selected attacks for our classifier agent, sole adversarial attacks such as the Fast Gradient Sign Method, Jacobian Saliency Map Attack, and Carlini-Wagner attacks can be applied to determine the performance of our models when trained with the aforementioned adversarial perturbations.

Chapter 2

Unsupervised Machine Learning for Cybersecurity Anomaly Detection in Traditional and Software-Defined Networking Environments

2.1 Introduction

With both the advent of the information age coupled with the rise of technological threats, there has been a greater emphasis on cybersecurity across several industries. Cyber attacks are becoming increasingly complex; several attacks have become even more stealthy in nature as they evolve to be undetectable by traditional, signature-based methods. Due in part to the growth of computing power, artificial intelligence (AI)/machine learning (ML) are anticipated to play critical roles in the future of cybersecurity. That is, the integration of AI/ML continues to be an important emerging technology area and will become increasingly critical to assist with making

complex decisions in the future within the cybersecurity domain. Recent and highly publicized cyber attacks within the decade include the infamous Stuxnet worm that inflicted damage on the Iranian nuclear facilities in Natanz, to the most recent and damaging attacks on the largest American oil pipeline, Colonial Pipeline [117]. In 2021 the National Academy of Engineering also designated securing cyberspace as one of the fourteen Grand Challenges in Engineering in the 21st Century, an indicator of more public focus towards the continuously growing field of cybersecurity and information assurance [82].

In addition to the growth of the cybersecurity field, and in large part due to the increasing expanse of the internet, several advances in the field of software-defined networking (SDN) have made network administration of devices such as routers and switches significantly easier [56]. Traditional network administration involved the manual configuration of network devices, usually by which a network technician had to physically interface with each device to update device configurations. Instead, SDN involves the utilization of an application programmable interface (API) in order to send configuration changes through a centralized controller device as shown by Figure 2.1. As opposed to a traditional network where the control and data planes both reside on an individual device, for SDN environments, there is a controller device that delineates the control plane which defines networking topology and forwarding of traffic. The data plane then performs the actual forwarding of data, and is reserved only for those networking devices [46]. Monitoring of SDN traffic involves the usage of OpenFlow, a special protocol that allows network administrators to analyze traffic on the data plane via a controller device residing on the control plane [25]. As a result, network administration tasks have been made significantly more manageable for any size of network infrastructure.

Albeit, while software defined networking has eased the burden of network administration, SDN is not without its flaws with respect to cybersecurity [25]. Software misconfiguration can pose a substantial threat to networking devices [108]. Network management devices used commonly in SDN have been subjected to

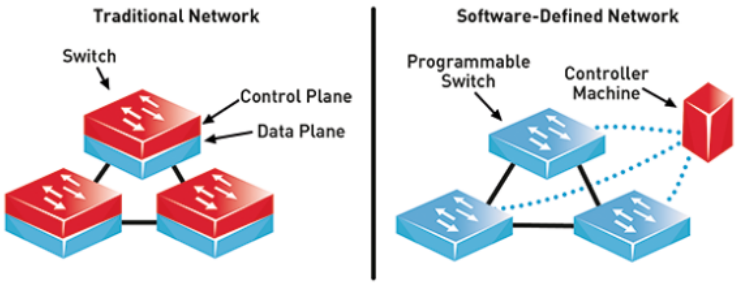


Figure 2.1: Traditional Networking Architecture versus Software Defined Networking [70].

cybersecurity attacks, such as one of the most recent cybersecurity attacks on the SolarWinds platform [8]. However, most network threats can still be discovered by the use of network intrusion detection systems (NIDS) [37].

Within cybersecurity, two primary forms of intrusion detection systems exist; these systems are either signature-based or anomaly-based. These NIDS that are signature-based often rely on a database of known threats, such as VirusTotal [71], in order to leverage existent information of threats. Oftentimes, for threats without a known signature, anomaly-based NIDS determine suspicious network traffic through the usage of statistical techniques and ML methods. Both signature-based NIDS and anomaly-based NIDS are crucial components to any cyber defense strategy.

While many signature-based NIDS can detect threats on an information network, due to the proliferation of cyber threats both modest and sophisticated and the lack of labeling in actual network packet captures, there has been a greater emphasis on the need of anomaly-based methods in cybersecurity. To the best of our knowledge, a majority of the literature concerning the anomaly-based intrusion detection systems, especially within the scope of an SDN environment, have involved the use of *supervised* learning techniques. These techniques necessitate the usage of labeled data for training. In reality, network traffic is unlabeled. Cybersecurity analysts do not have

sufficient time to analyze all network traffic and craft signatures for attacks in real-time, henceforth, there is a need to easily distinguish whether traffic is malicious or normal.

Most cybersecurity datasets have mainly included the capture of network flow data, among several others [94]. Network flow data contains a summary of packets that traverse a computer network and can be represented in a spreadsheet format. This is in contrast to packet capture data which provides a more granular overview of all the Ethernet packets sent over a network connection and is presented primarily in a .pcap, a packet capture, format. Both network flow and packet capture data can be obtained by a protocol analyzer software, such as Wireshark [119], tcpdump [41], or even through Cisco's proprietary NetFlow [22]. Popular network flow datasets seen in most cybersecurity research includes the KDD-Cup 99 dataset and the improved NSL-KDD dataset [114], the ISCX dataset [107], the UNSW-NB15 dataset [79], and the CIC-IDS2017 dataset [106].

Our goal for this study is to develop and implement robust anomaly detection models that can identify and mitigate anomalous internet traffic. We present three ideas for the field of unsupervised ML-network intrusion detection.

- First, we present the usage of the Restricted Boltzmann Machine (RBM), an unsupervised neural network developed and championed by [45] for feature engineering on the well-known KDD-Cup 99, NSL-KDD, CIC-IDS2017, and DNP3 intrusion detection datasets coupled with the one-class support vector machine (OCSVM), and the Isolation Forest (I-Forest) unsupervised learning models. We utilize the RBM both independently and for feature engineering in conjunction with aforementioned unsupervised learning models. This model is trained and tested in both a traditional networking environment and in a SDN environment that incorporates elements of supervisory control and data acquisition (SCADA) in conjunction with industrial internet-of-things (IIoT) devices operating as part of an industrial control system (ICS) utilizing the

distributed network protocol version 3 (DNP3). We further analyze an extension of the RBM, the Deep Belief Network (DBN) as one of our primary models in order to better capitalize of the hierarchical nature of the DBN to represent abstractions in the underlying network traffic.

- Second, we introduce the usage of a unique Generative Adversarial Network (GAN) method for anomaly detection independently and in conjunction with the RBM for feature engineering within the traditional networking environment and in the SCADA/ICS-SDN environment using the aforementioned DNP3 dataset. To the best of our knowledge, this novel approach in which an RBM is implemented in conjunction with the feature layer of our GAN-based anomaly detection approach as proposed initially by [4] has not yet been implemented in the cybersecurity literature, especially with respect to an SCADA/ICS-SDN environment. This novel GAN allows for the training of both a generator and discriminator in order to better classify anomalies; in this context for network traffic. Our hypothesis is that the training of our discriminator using samples generated will lead to better results for classification of anomalous traffic.

- Third, in addition to the GAN model presented with other unsupervised approaches, we also present these models across datasets to determine the generalizability of our models. Training and testing across datasets can provide insight on the performance of an ML-based algorithm when presented with data not found in training, similar to zero-day attacks in cybersecurity.

In addition to the realism by applying *unsupervised* models as network data is in raw form is unlabeled naturally, we expect that through our study we may provide further understanding on model performance against potential zero-day attacks that have no known signature. Henceforth, our unsupervised models create an ML-intrusion detection system that can effectively detect anomalies.

The rest of the paper is structured as follows: In Section 2.2 we provide an overview of relevant literature in network intrusion detection systems for cybersecurity. In

Section 2.3, we discuss the methodology used in this study, to include the dataset used along with the ML models employed within the study. Section 2.4 discusses the numerical results of our ML methods across our models using predefined metrics. Finally, section 2.5 provides the conclusion and examines future work.

2.2 Literature Review

Due to the ever-increasing expansion of the internet, along with the increasing efficiency of computing systems, there has been a rise in applying sophisticated ML methods within the cybersecurity literature. One of the more common supervised approaches, the usage of the support vector machines (SVM) has been implemented on the earlier KDD-Cup 99 dataset [59, 57] and on the more recent NSL-KDD dataset [87]. The SVM proposed in both studies involving the KDD-Cup 99 yielded relatively accurate results, with 89.85% in the study presented by [59] and 95.11% in [57]. [57] also noted that in addition to their high SVM accuracy, they obtained a low false-positive rate of 0.8%. High accuracy with moderately high F1-score performance was obtained in [87]. SVMs have also been combined with other ML models, including random forests (RF) for intrusion detection, as described by [126]. Additionally [1] implemented several machine learning techniques including the SVM, k-Nearest Neighbors, and RF which provided the basis for detection of DDoS attacks combined with feature engineering techniques such as feature selection and feature normalization. In addition to the usage of the machine learning techniques for network flow analysis, cybersecurity researchers benefited from the implementation of deep learning through the usage of supervised neural networks [118].

The application of neural networks has been a source of recent interest in the cybersecurity field. One of the first cybersecurity studies to employ deep neural networks within an SDN environment was by [112]; the researchers achieved 75.75% accuracy when selecting six features commonly passing through an SDN controller. Gated Recurrent Unit-Recurrent Neural Networks (GRU-RNN) improves upon the

traditional neural network and provides better processing of time-series data; a GRU-RNN model achieved 89% on accuracy, an 87% on precision, and 89% on recall, and an 88% on F1-score [113], bettering the performance seen in [112]. GRU in conjunction with transfer learning was applied in [109]; the proposed approach achieved an exceptionally high accuracy of 99.92% on the KDD-Cup 99 dataset and 94.22% on the UNSW-NB15 dataset. [5] also applied Deep Neural Networks (DNN) and convolutional neural networks (CNN) in conjunction with long-short term memory (LSTM) for DDoS detection and found their CNN to yield the best accuracy in detecting malicious traffic, with performance upwards of 99%. A notable neural network-type model, the RBM, also has been employed within the cybersecurity field. RBMs are unique in that they can provide feature engineering and can perform in conjunction with other ML techniques. [101] employed the RBM in concert with other supervised models including the SVM, random forest, and decision tree. Their study suggested the random forest outperformed all other models when paired with the RBM. DBNs extend upon RBMs; these models stack several RBMs in order to provide better performance. DBNs can be used for feature extraction in conjunction with other models which can yield better performance compared to the model alone [104]. A comparison of supervised machine learning methods for intrusion detection to determine each model's strength in dataset generalization was presented in [31], the authors noted that tree-based methods performed best with respect to other supervised machine learning methods.

While supervised methods have been heavily employed in the field of anomaly detection in cybersecurity, there has been less of an emphasis on employing unsupervised learning techniques in the cybersecurity domain. Unsupervised techniques in the cybersecurity domain could be extremely useful as most network traffic is not labeled. Classification of network traffic, to determine whether traffic is benign or malicious when an alert is sent from an IDS, is usually the task of the cybersecurity analyst. Two common ML models, the SVM and Random Forest, both have unsupervised counterparts. One-class support vector machines (OCSVM), an

unsupervised extension of the support vector machine, are a popular unsupervised method in cybersecurity research. Cybersecurity researchers have applied OCSVMs with several kernels, including both linear and polynomial [26] and radial basis function [75]. [75] also utilized an autoencoder for feature extraction with the OCSVM, while in [20] a convolutional autoencoder was the model chosen for the analysis of the NSL-KDD dataset. Isolation Forests (I-Forest), a recent unsupervised learning method that detect anomalies using tree-like structures, have been employed in the cybersecurity domain [74, 103]. Sadaf and Sultana furthered the Isolation Forest by combining the method with an autoencoder across those same four metrics. Sadaf and Sultana's performance on the I-Forest yielded much better results: accuracy was at 95.40%, precision was 94.81%, recall was 97.25%, and F1-score was 96.01%. [103] akin to the aforementioned studies, also used an autoencoder for analysis. Additionally, unique multiresolution analysis provides a potential means for efficient intrusion detection. [63] provided an anomaly detection mechanism inspired by the human immune system and obtained accuracies of 97.37%, 99.97%, 99.56%, and 99.75% on the NSL-KDD, UNSW-NB15, CIC-IDS2017 and CSE-CIC-IDS2018 intrusion detection datasets.

RBMs, as mentioned earlier, can both be applied in a supervised context when paired with supervised models, and can also be utilized independently as a standalone unsupervised model. [7] utilized both contrastive divergence and persistent contrastive divergence on another cybersecurity dataset, the ISCX dataset and determined that the persistent constrastive divergence provided the best results when used with a normalized, balanced dataset [7]. [29] improved upon the previous study with the same dataset and included a novel method of RBM training, quantum anneling, against contrastive divergence. Across these methods, Constrastive Divergence produced the best results, but the novel quantum annealing training method was still effective when trained on the normal, unbalanced dataset [29]. [28] also opted to apply the stacked-RBM method, the DBN, on the NSL-KDD

dataset. Their DBN obtained an accuracy of 71.90% but gained a high true positive rate of over 90%.

The usage of GANs, first purposed by [40], normally accept input in a form of an image, but for our study will be applied towards cybersecurity for network traffic classification. GANs serve as an adversarial method in which intrusion detection and prevention systems can be subverted. GANs intend to reduce the performance of other ML models as the model intends to supply deceptive input to a classification model. [128] employed GANs in comparison with other ML models on the NSL-KDD network flow dataset and found that the detection rate against their proposed ML models dropped precipitously; the evasion rate after applying the GAN was over 99%. Wasserstein GANs (WGANs), introduced by [42], were also tested on the NSL-KDD on artificial neural networks [64] and convolutional neural networks (CNNs) [127]. [64] found that when the WGAN was applied for classification in their neural network, the model significantly struggled with discerning input, dropping from 70% detection to less than 1% on their detection rate. [127] also found that the true positive rate of their CNN dropped to 47.6% after achieving a high true positive rate of 97.3% when not trained in an adversarial context. An overview of the recent literature concerning the application of ML towards network intrusion detection is shown in Table 2.1.

From our literature review, there appears to be a dearth of the application of GAN-based methods as a means of improving anomaly classification in network traffic, only indicating as to how effective GANs are at "fooling" machine learning models. As a direct consequence, in our study, we intend to use a GAN-based approach that utilizes adversarial learning which possesses limited assumption of the model architecture, as a classifier for anomaly detection. We aim to determine whether adversarial training, in this context for our GAN-based method, assists with classifying threats on an information network.

Table 2.1: An overview of selected literature concerning the application of ML towards network intrusion detection (OC)SVM: (One Class) Support Vector Machine, RBM: Restricted Boltzmann Machine, DBN: Deep Belief Network.

Model (Author)	Dataset	Features	Metrics	Performance		
SVM [57]	KDD-Cup 99	All Features	Accuracy False Positive Rate	95.11% 0.8%		
SVM [87]	NSL-KDD	All Features 3-Feature Subset	Accuracy F1-Score	99% (All Features) 91% (3-Feature Subset) 0.77 (F1-Score)		
Deep Neural Network [112]	NSL-KDD	SDN 6-Features Subset	Accuracy	75.75%		
Recurrent Neural Network [113]	NSL-KDD	SDN 6-Feature Subset	Accuracy Precision Recall F1-Score	89% 87% 89% 88%		
Restricted Boltzmann Machine with SVM, Random Forest, Decision Tree [101]	NSL-KDD	16-Feature Subset	Accuracy Precision Recall F1-Score	RBM-SVM 76.81% 81.03% 79.21% 76.70%	RBM-Random Forest 77.8% 81.33% 79.54% 76.94%	RBM-Decision Tree 76.86% 81.22% 79.35% 76.74%
Restricted Boltzmann Machine [7]	ISCX	16-Feature Subset	Accuracy True Positive True Negative	Contrastive Divergence 88.6% 88.4% 88.8%	Persistent Contrastive Divergence 89.0% 84.2% 93.8%	
Restricted Boltzmann Machine [29]	ISCX	All Features	Accuracy False Positive Rate Area Under Curve	98.84% 3.44% 98.84%		
Deep Belief Network [104]	NSL-KDD	All Features	Accuracy	SVM 88.33%	DBN 89.54%	DBN-SVM 92.84%
Deep Belief Network [28]	NSL-KDD	All Features	Accuracy True Positive False Positive	71.91% 93.70% 41.53%		
Radial Basis Function OCSVM with Autoencoder [75]	CICIDS2017	19-Feature Subset	Accuracy Precision Recall F1-Score	99.35% 99.97% 98.28% 99.11%		
Polynomial OCSVM Linear OCSVM Isolation Forest [26]	NSL-KDD	Subset of Selected Features	Accuracy False Positive Rate Precision Recall F1-Score	Polynomial: 86.3% 7.5% 86.6% 78.0% 82.9%	Linear: 86.3% 7.7% 88.4% 78.2% 83.0%	Isolation Forest: 89.9% 7.9% 89.0% 85.5% 87.2%
Isolation Forest [74]	NSL-KDD	All Features	Accuracy Precision Recall F1-Score	85% 45% 80% 55%		
Isolation Forest with Autoencoder [103]	NSL-KDD	All Features	Accuracy Precision Recall F1-Score	95.40% 94.81% 97.25% 96.01%		

2.3 Methods

2.3.1 Data

In this section, we cover the three traditional networking datasets in addition to our SCADA/ICS-SDN dataset used in the study replicating network traffic that can be captured using a network protocol analyzer, such as Wireshark or tcpdump, or through the usage of a NetFlow data which provides a summary of network traffic or analysis [22].

KDD-Cup 99 Dataset. The KDD-Cup-99 dataset was first proposed by the ACM Special Interest Group on Knowledge Discovery and Data Mining co-opted with the Defense Advanced Research Projects Agency (DARPA) and the Massachusetts Institute of Technology [114]. The dataset is comprised of 494,021 individual network packets spread across 41 different features with the network traffic comprised of five different types of traffic which are described below

- Normal traffic
- Denial-of-Service (DoS): An attack which disrupts the ability to communicate or interact with a computing system.
- Probe: An attack where an hacker attempts to determine open ports and services hosted on the network, primarily used for reconnaissance activities prior to more thorough attacks.
- User-to-Root (U2R): Traffic that is reflective of when an unauthorized user successfully achieves administrative privileges.
- Remote-to-Local (R2L): Attack traffic indicative of an unauthorized external computer successfully gaining access to a computer on the local area network.

NSL-KDD Dataset. In addition to the original KDD-Cup 99 dataset, we opt to use the NSL-KDD dataset for our study. The NSL-KDD dataset improves upon

the KDD-Cup 99 dataset by removing several redundant records in the dataset. Akin to the KDD-Cup 99 dataset, the NSL-KDD dataset has a total of five different types of traffic with 41 different features. The NSL-KDD dataset also possesses a greater taxonomy of threats. The dataset is divided into a train dataset with a total of 125,973 network packets and a test dataset with 22,544 packets [114].

DNP3 Dataset. In addition to applying an emulated SDN dataset using the NSL-KDD dataset, we also apply our analysis on the novel DNP3 Intrusion Detection Dataset as crafted in [52]. This dataset was constructed in the context of SDN technology, specifically using SDN-microsense technology which aims to provide a robust and resilient cyber infrastructure for IoT devices. The dataset was devised through the implementation of the Distributed Network Protocol 3 (DNP3) protocol. DNP3 is a widely used communication protocol for integrating various types of data acquisition and control devices in industrial automation and control systems. It is commonly used in the electric utility, oil and gas, and water/wastewater industries. The protocol provides functions for device configuration, data acquisition, and control, and includes features such as data integrity checking and time synchronization. The DNP3 dataset was crafted through the implementation of the supervisory control and data acquisition (SCADA) infrastructure consisting of master terminal units serving as the controller device and several remote terminal units serving as the industrial control system (ICS) devices. Also on the network, attacker platforms performed a wide variety of attacks on both the master and remote terminal units. Some attacks with respect to the Distributed Network Protocol 3 arise due to improper functional codes either being sent to the controller device or the outstation. The types of network traffic, including attacks, are described below:

- Normal traffic
- Disable Unsolicited Messages: This attack disables the ability of remote terminal units sending messages to the master terminal units. This attack aims to disrupt the functionality of notifying the master terminal unit of any issues

or abnormalities on the device by sending a packet with the DNP3 function code 21, which requests to disable all the unsolicited messages on the target.

- Cold Restart: Cold restart attacks forces the remote terminal unit to perform a full restart, leaving the device unavailable to be accessed for a time. This attack is a form of a DoS attack.

- Warm Restart: Warm restart attacks only prompts the remote terminal unit to restart the DNP3-based applications only. This attack still is a form of a DoS attack.

- Discovery: Akin to network probe attacks, the discovery attack attempts to determine open ports and services that are hosted on the network.

- Initialize Data: This attack attempts to subvert the integrity of the remote terminal unit by forcing the outstation to revert back to default values. This attack occurs through the misuse of function code 15, which instructs the outstation to reinitialize the original device settings.

- Stop Application: The stop application attack effectively halts all communication between the master terminal unit and the remote terminal unit in a form of a man-in-the-middle (MITM) attack. This attack is conducted via a malicious device transmitting the function code 18 which corresponds to the stop application code. This prevents the outstation from communicating anymore with the master station..

- Replay: Akin to the stop application attack, the replay attack applied an MITM attack in order to intercept and delay transmissions of packets originating from the master terminal unit sent to the remote terminal unit.

SDN Dataset Emulation. Both the aforementioned KDD-Cup 99 and NSL-KDD datasets utilized the NetFlow monitoring software or similar monitoring software for their collection. As opposed to NetFlow which is usually collected via

a networking device such as a router or a switch, SDN traffic is collected via the OpenFlow protocol [25]. Instead of directly applying the OpenFlow protocol for collection of our SDN traffic, we opt to emulate the SDN traffic by extracting six features present in SDN OpenFlow traffic as presented in [112] and [113] from both the KDD-Cup 99 and NSL-KDD datasets. These selected features best simulate network flow traffic over an SDN controller device. These six features include the *duration, protocol_type, src_bytes, dst_bytes, count,* and *srv_count*. These features are easily collected by both NetFlow and OpenFlow protocols and provide a means to emulate SDN traffic. These features are compared alongside the full dataset to determine the effectiveness of classifying threats even when presented with a limited set of features. In addition to selecting these six features, we also analyze the DNP3 dataset which was constructed specifically using SDN-microsense technology.

CIC-IDS2017 Dataset The CIC-IDS2017 dataset is a recently generated intrusion detection dataset crafted by the Canadian Institute of Cybersecurity (CIC). This dataset was generated over a 5-day period and was intended to accurately represent many modern organizational networks. The dataset comprises of a variety of attacks [106]. The dataset has 15 total traffic classes which can be roughly divided among six different types of attack classes. These types of attacks subdivided by class are described below [36]:

- Benign: Normal Network Traffic
- Denial of Service: DoS attacks types present in the CIC-IDS2017 include DDoS, DoS Slowloris, DoS Hulk, DoS GoldenEye, and Heartbleed
- PortScan: Akin to probing attacks, this attack intends to discern open ports and services. The CIC-IDS2017 dataset has one type of attack of this class.
- Bot: Attack traffic generated by bots. There is one type of attack present in the CIC-IDS2017 dataset covering bot attacked, labeled Bot.

- Brute-Force: These attacks attempt to gain access to a network resource through repeated attempts. The FTP-Patator and SSH-Patator attacks present in the CIC-IDS2017 dataset represent brute force attack traffic.

- Web Attack: These attacks occur against web applications and web browsers. Attacks present in the CIC-IDS2017 which are web-based include the brute force web attack, Cross-Site Scripting (XSS), and SQL Injection.

- Infiltration: Infiltration involves an adversary successfully gaining unauthorized access to a network resource. There is one attack present in the CIC-IDS2017 dataset covering infiltration.

Dataset Preprocessing Our traditional networking datasets to include the KDD-Cup 99, NSL-KDD, and CIC-IDS2017 datasets have a greater proportion of normal traffic as opposed to attack traffic. In order to provide a more balanced dataset, and to accommodate for anomaly detection, we instead opt to group the attack traffic into a single anomaly class. With this preprocessing step, we obtain a more balanced dataset that we utilize for our analysis. As opposed to our NSL-KDD and KDD-Cup 99 datasets, [52] has provided a balanced dataset; this balance is across all classes of attack traffic and normal traffic. Each class of traffic, whether normal or one of the attacks, has an equal amount. The ratio between normal traffic to attack traffic on the balanced dataset is one to seven, which then presents an issue for anomaly detection. Thus, in order to provide an improved model training process, we utilize synthetic minority oversampling technique (SMOTE) as defined in [18] to provide balanced classes between normal and anomalous (attack) traffic.

2.3.2 Models

In this section, we discuss our primary methods of analysis, the Generative Adversarial Network, Restricted Boltzmann Machines and its extension of Deep Belief

Networks, and other recent ML models that are used in conjunction with our primary method of analysis as a baseline for comparison:

Generative Adversarial Networks. Generative Adversarial Networks (GAN) were first proposed by [40] in 2014. GANs are an unsupervised, generative deep learning model originally developed with the specific goal of producing artificial but realistic images. GANs consist of two sub-networks denoted as the generator and discriminator which are in competition with each other. The learning process of the generator attempts to produce images that appear realistic, while discriminator has the purpose of attempting to distinguish the artificial image from the real image. The generator possesses a decoding network structure that tries to learn the distribution of input data from the latent space. The discriminator serves as as a classifier that determines whether the images it receives are real or artificial. The goal of the generator during training is to create errors in the classifying process of the discriminator (obtained by deceiving the discriminator by producing images that the discriminator cannot detect as artificial). The two sub-networks, the generator network G and the discriminator network D are trained simultaneously through a two-player minimax game.

The purpose of the discriminator D during training is to maximize the likelihood that real training images will be correctly identified as "real" while the artificial image will be correctly identified as "fake." The generator network utilizes training based on the ability of the discriminator to deceive the generator network by minimizing or by equivalently maximizing. The GAN design is described in Figure 2.2. Due to training instability on the GAN however, experimental methods to solve this problem have been introduced into the literature. One of the most effective ways to overcome training instability has been proposed by [92]. Their solution, the Deep Convolutional GAN, is another GAN in which the convolutional layers and batch normalization are joined by fully connected layers.

Anomaly Detection Using Generative Adversarial Networks. Akcay et al. introduced the GAN-based anomaly detection method used to further the

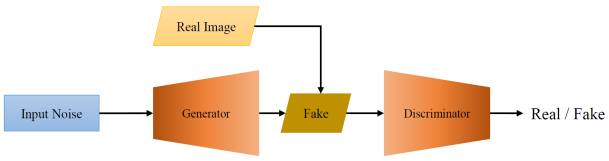


Figure 2.2: Traditional GAN Architecture, adapted from [4].

GANs, referenced as GANomaly [4]. The model proposed by Akcay et al. further improves GANs by introducing a new architecture for the generator. The GAN-based anomaly detection method consists of the following three sub-networks: an autoencoder network that functions as a generator (G_E and G_D), E as the encoder network, and the discriminator network D, as described in Figure 2.3.

The generator is responsible for learning the distribution of the data that is non-anomalous and performing the reconstruction of the real image through the usage of the autoencoder network. The network of the generator works by first having the encoder section, G_E obtaining one sample image $x \in \mathbb{R}^{c \times h \times w}$, where c, h, and w are the channels, height, and width of the image, respectively. Following obtaining the first sample, the encoder then sends the sample image to its architecture consisting of convolutional layers followed by batch-normalization and Leaky-Rectified Linear Unit (ReLU). The encoder network then reduces the input image x to the vector $z \in \mathbb{R}^d$, which usually possesses a lower dimension than x which also has the best representation of x in the latent space. The decoder section, G_D, then is responsible for projecting the z vector to the input image space and reconstructing the input image, thus denoted as $x' \in \mathbb{R}^{c \times h \times w}$. The network has a deep convolutional GAN architecture consisting of convolutional transpose layers, Leaky-ReLU, activation functions, and batch-normalization with a tanh layer. Henceforth, the autoencoder obtains an image x and then outputs the reconstructed image $x' = G_D(z)$, where $z = G_E(x)$ [4, 93].

The encoder network E returns the artificial image x' back to the latent space in order to find the extracted features of x' in the latent space where $z = E(x') \in \mathbb{R}^d$.

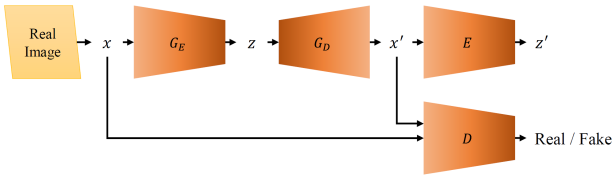


Figure 2.3: GAN-based Anomaly Detection Architecture, adapted from [4].

The vector z' has dimensions similar to z and can be directly compared and allows for the calculation of losses. Minimizing the distance between z and z' also assists the generator network to better learn the distribution of real images. Finally, the discriminator network works akin to a traditional GAN in that the discriminator network works as a classifier, with the goal of distinguishing between artificial and real examples. During the training, real sample images $x \in \mathbb{Q}$ and the corresponding reconstructed image x' are then fed to the discriminator. The discriminator network then learns whether if these images are real or artificial. To optimize GAN-based methods, the parameters of all three encoder networks (G_E, E and D) and the decoder network (G_D) need to be learned. In general, the sub-networks E and D have a structure similar to the G_E, the generator-encoder network, but it has its own unique weight and parameters. We selected the following network architecture based upon our initial analysis: For our encoder networks, we opt to utilize three convolutional layers which are then followed by batch-normalization and the Leaky-ReLU activation function. Sequentially in our architecture, the decoder network consists of three convolutional transpose layers, batch-normalization with the Leaky-ReLU activation function, and a tanh function in the last layer [4, 93].

Previous works applying GANs in the sense of intrusion detection only implemented these models as a means to generate traffic intended to fool an already trained IDS, akin to the work seen in [127] and [65]. These works, while effective at surpassing trained defenses, do not instead implement GANs as a means to better train a classifier network. Our GAN-based detection method instead allows our discriminator network serving as an IDS to obtain adversarial samples in order to improve its training; by

exposing the classifier to adversarial samples, it can better determine a baseline for normal traffic as demonstrated in works such as [93].

Restricted Boltzmann Machines. Restricted Boltzmann Machines (RBMs) are special forms of stochastic artificial neural networks in which the neurons have the ability to be in a state that is either on or off. RBMs are derived from an earlier physics-based model, the Boltzmann Machine [45]. Boltzmann Machine and by extension RBMs consist of two primary layers; these layers are a "hidden" layer and a "visible" layer. [45] created the RBM to control the energy of Boltzmann Machines by only having connections between hidden neurons and visible neurons [100].

For $v = (v_1, ..., v_m)$ representing M amount of visible nodes, $h = (h_1, ..., h_n)$ representing N amount of hidden nodes and for $v_i, h_j \in \{0, 1\}$, we can construct an energy function as follows: [45, 100]

$$E(v, h|\theta) = -\sum_{m=1}^{M} a_m v_m - \sum_{n=1}^{N} b_n h_n - \sum_{m=1}^{M} \sum_{n=1}^{N} v_m h_n w_{mn},$$

θ, a_m, b_n, and w_{mn} for this energy function represents the RBM parameters. a_m and b_n represent the terms of bias for the visible and hidden layers. Additionally w_{mn} indicates the weights for the two different visible and hidden layers. This energy function provides a probability distribution for the states of the hidden and visible neurons within the RBM [45, 100].

Using this energy function, the equation for probability is then constructed as follows: [45, 100]

$$P(v, h) = e^{-E(v,h)} / \sum_{m,n} e^{-E(v,h)},$$

The energy function is essential in the training process of the RBM; the energy function provides for the computation of the gradient of the training data log-likelihood with respect to the biases a_m and b_m and weights w_{mn} of the network. The aforementioned energy function thus provides a lower energy to states corresponding with a higher probability. Using the above energy function, a "partition function"

is then constructed by the following equation $\sum_{m,n} e^{-E(v,h)}$. This function is a normalization factor provided through the summation of both visible and hidden elements. We can then utilize the energy function in order to then provide the probability for both the hidden and visible elements. Following such, the visible elements v are assigned a probability through construction of the following equation using the above-mentioned energy function: [45, 100]

$$P(v) = \sum_n P(v,h) = \frac{\sum_n e^{-E(v,h)}}{\sum_m \sum_n e^{-E(v,h)}}.$$

The energy function can also be used to assign the probabilities to the hidden elements. Akin to the visible elements, we can then assign the hidden elements, h, probability through the following equation [45, 100]:

$$P(h) = \sum_m P(v,h) = \frac{\sum_m e^{-E(v,h)}}{\sum_m \sum_n e^{-E(v,h)}}.$$

As opposed to other neural networks that utilize gradient descent during training in order to determine an optimal loss value, RBMs adhere to a two step process for training, detailed below:

Gibbs Sampling The Gibbs Sampling step is the initial step in the training process for the RBM. From v, an input vector, the binary state probability of the hidden values h_j being equal to 1 indicating an "on" state is provided via the equation: [45, 100]

$$P(h_j = 1|v) = 1/(1 + e^{b_j + \sum_i v_i w_{ij}}).$$

Contrastive Divergence After the Gibbs sampling is performed, contrastive divergence updates weights for the RBM. Contrastive Divergence calculates the input vectors probabilities (with input vectors being denoted v_0 and v_n) which are then calculated from the utilization of the probabilities of the given activation for h_0 and h_n. In summary, constrastive divergence provides an approximation of the gradient of the log-likelihood of the input data. Contrastive divergence applies a Markov chain for the approximation via the Gibbs sampling process. The equation for calculating

the probabilities of the input states when using the contrastive divergence step is represented below: [45, 100]

$$P(v_i = 1|h) = 1/(1 + e^{a_i + \sum_j h_j w_{ij}}).$$

After the identification of the the hidden unit binary states, v_i is assigned to the value of 1. With that step, the "reconstruction" occurs. This steps thus provides the updates in the network weights: [45, 100]

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle)_{data} - \langle v_i h_j \rangle_{reconstruction}).$$

Persistent contrastive divergence (PCD) is another method for the training of RBMs. During PCD, instead of re-initializing the Gibbs sampler during each iteration, the final state initializes the concurrent Gibbs sampler step [45, 100].

Deep Belief Networks Deep Belief Networks (DBNs) are a popular extension of the aforementioned RBMs. DBNs are constructed via sequentially stacking RBMs. Training of DBNs follow that of RBMs, the primary difference is the "stacked" nature wherein the output of one RBM is directly fed into the input of the next RBM. This structure in turn creates a more hierarchial model with several layers. As a result, DBNs can better model more abstract representations of data which can lead to better performance [49].

One-Class Support Vector Machine. The One-Class Support Vector Machine (OCSVM) is an unsupervised ML algorithm that intends on classifying data into two distinct groups, normal and anomalous, through the usage of a hypersphere. Anomalies are then distinguished from the normal data via the spherical boundary in the feature space. The objective is to find the minimum volume of the hypersphere through solving the optimization problem:

$$min_{R,a} R^2 + \frac{1}{v} \sum_{i=1}^{n} \xi_i,$$

$$\text{Subject to: } ||x_i - a||^2 \leq R^2 + \xi_i \text{ for all } i = 1, 2, 3, ..., n,$$
$$\xi_i \geq 0 \text{ for all } i = 1, 2, 3, ..., n,$$

where ξ serves as a slack parameter and $v \in [0,1]$ determines the percentage parameter estimation; when v is small, more data can be present in the hypersphere. When v is larger the size of the hypersphere is reduced to serve as the penalty parameter. The larger the hypersphere, the more tolerant the OCSVM model is towards outliers. The smaller the hypersphere, the more sensitive the model is to outliers [115].

Isolation Forest. The Isolation Forest (I-Forest) is another unsupervised ML algorithm that recursively creates partitions for data, and determines anomalies based upon the total amount of partitions. The amount of partitions serve as a measure of anomaly for data. Fundamentally, there are fewer partitions that are needed in order to isolate anomalous data as opposed to normal data. The I-Forest works by employing several smaller trees to identify every data point such that non-anomalous data is at the lowest level of the trees at the leaves, while anomalous data resides closer to the root of the tree [66]. The algorithm for the isolation forest is described below:

Let $h(y)$ describe the amount of iterations needed to isolate the sample y in tree t_i. Thus the average amount of iterations needed to isolate a sample y can be calculated as:

$$h(y) = \frac{1}{T} \sum_{i=1}^{T} h_{t_i}(y).$$

Since $h(y)$ is reliant on the amount of samples in dataset (n), the anomaly score once normalized, $s(y, n)$, can be calculated as:

$$s(y, n) = 2^{\frac{-h(y)}{c(n)}}, \text{ where } c(n) = 2H(n-1) - \frac{2(n-1)}{n}.$$

Note that $H(n)$ serves as the harmonic number. As $h(y)$ approaches 0, $s(y, n)$ approaches 1, indicating that the sample y is more apt to be an anomalous sample. Inversely, as $h(y)$ approaches $(n-1)$, $s(y, n)$ approaches 0, indicating that the sample is likelier to be normal [91].

In this work, we opt to examine our unsupervised learning models across a variety of intrusion detection environments. To the best of our knowledge, an analysis

comparing these unsupervised learning models across both traditional and software-based IoT networking environments has not yet been accomplished. Thus, our work provides a robust examination of not just our base unsupervised learning models, but also ensemble models for effective network anomaly detection.

2.3.3 Experimental Design

We opt to apply five different experiments within this study. These experiments will analyze the effectiveness of using a RBM for feature engineering, in addition to determining performance with only those features present in an SDN environment. Our five experiments are described in the following list below:

- Within each dataset (i.e., training and testing on the same dataset):
  - **Experiment W-I:** We execute our five main models, OCSVM, IF, RBM, DBN, and GAN, when using all features solely on the NSL-KDD and KDD-Cup 99, and CIC-IDS2017 datasets. This allows us to benchmark the four models against each other and determine which model performs the best when applied within a traditional networking environment.
  - **Experiment W-II:** We execute our five main models when using only the six features present in an SDN environment for the NSL-KDD and KDD-Cup 99 subsets alongside the full DNP3 intrusion detection dataset. This allows us to benchmark the four models against each other within the context of an SDN environment and examine the extent of model performance when operating in an infrastructure reflecting an SDN environment.
  - **Experiment W-III:** We pair RBM with our other four main models when using all features across all datasets, namely, RBM+OCSVM, RBM+IF, RBM+DBN and RBM+GAN. This allows us to examine if

feature engineering through RBM improves model performance. For our DBN model, we increase the amount of RBMs in the network by one.

- Across datasets (i.e., training on one dataset and testing on the other dataset):
  - **Experiment A-I:** We execute our five main models OCSVM, IF, RBM, DBN, and GAN, instead across the NSL-KDD and KDD-Cup 99 datasets. This allows us to examine the degree of generalizability of the models to external datasets.
  - **Experiment A-II:** We apply the RBM for feature engineering across the OCSVM, IF, DBN, and GAN models using the aforementioned NSL-KDD and KDD-Cup 99 datasets to determine whether the RBM improves the generalizability of the model when applied to external datasets.

2.3.4 Model Evaluation

Training Process Our hypothesis is that our unsupervised neural network and deep learning models proposed, both the RBM and the GAN-based model, will be more successful in discerning threats within network traffic due to abnormalities within their network traffic. With respect to the RBM, DBN, OCSVM, and the I-Forest, all of these models utilize a training dataset that consists of both normal traffic and anomalous traffic for training. Our GAN-based approach instead however is trained on non-anomalous data; the generator's parameters do not provide the training for the generation of the anomalous features of x. The reconstructed image x' has additionally lost the presence of its anomalies, as z', which is its latent representation outputted by the E, additionally lacks anomalous features. This change thus allows for the observation of dissimilarity between both the reconstructed image x' and the anomalous image x, and between the latent vectors z' and z. For each dataset used in our study, the training dataset was separated into ten distinct datasets with differing amounts of anomalies present for cross validation. If the anomalies present a larger

than a threshold, we hypothesize that the model will be able to discern the anomalous traffic from the normal traffic. If the anomalies are lower than a certain threshold; however, the model might not be able to discern the anomalies as effectively.

Testing Process Compared to the training process, our GAN-based model is able to accept input with anomalies for testing purposes. To accurately determine the anomaly detection performance on our GAN-based model, we calculate the anomaly scores for all test samples. In order to detect anomalies in the network traffic, we use the threshold value in conjunction with the scaled anomaly scores. To determine the threshold value, we first take the ready testing dataset that has anomalies present in the network traffic. The trained generator reconstructs network traffic to obtain the scaled anomaly score based on the testing dataset, and then we determine the optimal threshold on the scaled anomaly scores such that the model's performance is maximized on the specific evaluation metric of interest. Our training and testing process is described in Figure 2.4. Akin to the training process as well, the testing dataset is divided into ten datasets with different levels of anomalous network traffic and the results are compiled with respect to the model performance metrics.

2.3.5 Model Performance Metrics

To evaluate the performance of our various ML models, we employ metrics of accuracy, precision, recall, 1 - False Positive Rate (FPR), F1-score, and Geometric Mean (G-Mean). Accuracy is calculated as $\frac{TP+TN}{TP+TN+FP+FN}$, precision is calculated as $\frac{TP}{TP+FP}$, recall is calculated as $\frac{TP}{TP+FN}$, 1-FPR is calculated as 1 - $\frac{FP}{FP+TP}$, F1-Score is calculated as $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$, and G-Mean is calculated as $\sqrt{\frac{TP}{TP+FN}\frac{TN}{FP+TN}}$. TP, true-positive, serves as the metric accurately identifying anomalous network traffic as anomalous, FP is inaccurately identifying non-anomalous network traffic as anomalous, TN is accurately detecting non-anomalous network traffic as non-anomalous, and FN is inaccurately identifying anomalous network traffic as non-anomalous.

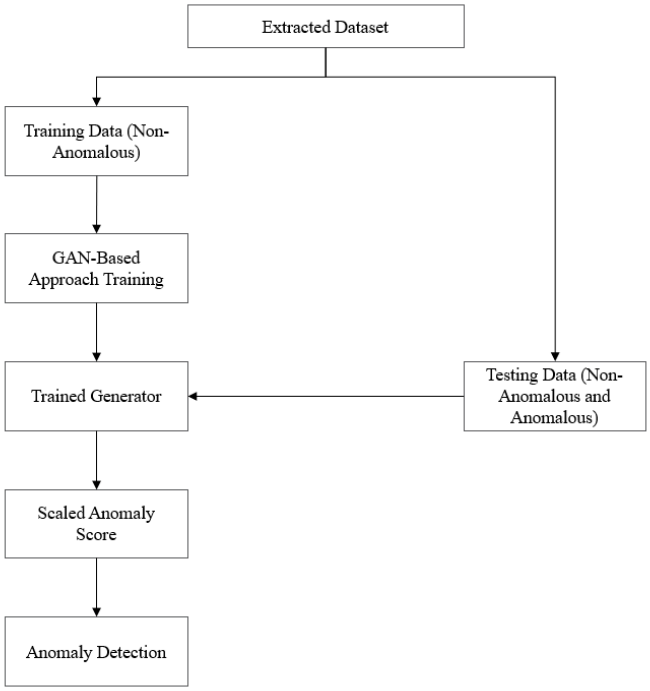


Figure 2.4: Training procedure versus testing procedure of our GAN-based model.

Accuracy aims to measure the ability to correctly classify anomalous and non-anomalous samples. Noteworthy is that many anomaly detection applications have a much smaller number of anomalous samples than that of non-anomalous samples. Henceforth, the accuracy metric could be potentially biased against the non-anomalous samples; thus, the metric cannot fully describe the ability of detecting anomalies. Therefore, we also consider the metrics of precision, recall, false positive rate, and F1-score. The precision metric takes into consideration correctly classifying true positives against all samples identified as positive. Likewise, the recall takes into consideration correctly classifying the true positives against all the positive samples. The false positive rate provides a metric to determine the amount of traffic that has been incorrectly classified as an attack when in actuality the traffic was benign. The F1-score takes into account both the precision and recall metrics by taking the harmonic mean of both metrics. We also consider the geometric mean which measures the ability of a model in balancing both false positive and false negatives samples.

After considering the various aforementioned metrics, we will determine the most effective model based upon the geometric mean.

2.3.6 Identification of the Best Model

To determine the best out of the models presented, we opt to first apply an analysis of variance (ANOVA) test [102]. If there are any statistically significant differences, we then apply a paired t-test to determine the best performing model in the experiment. As we are comparing within groups, the Bonferroni correction is applied [102]. The Bonferroni correction, $\frac{\alpha}{n}$ (where α and n denote the significance level and number of groups, respectively), changes the significance levels for the t-test to reduce the chances of a false positive when testing between groups. When there are four groups (such as in experiments W-I, W-II, and A-I) the significance level changes to $\alpha = 0.125$, and when there are three groups (such as in experiments W-III and A-II), the significance level changes to $\alpha = 0.167$. The best results are indicated in bold, while statistically significant results have either one or two asterisks for significant at α at a 5% or 1%, respectively.

2.3.7 Hyperparameter Analysis and Selection

Each of our machine learning models have distinct hyperparameters that affect the performance of the models. With respect to the OCSVM, the key hyperparameters considered in our study were the kernel function and the width of the radial basis function (gamma). We found that the best performance out of all kernel functions was through the usage of the radial basis function kernel. The width of the radial basis function (gamma) controls the influence of individual training examples, with a smaller gamma leading to a softer decision boundary and a larger gamma resulting in a more complex decision boundary. Through cross-validation, we found the best gamma at a small value of 0.001 yielded the best performance.

In addition to the hyperparameters present in the OCSVM, there are also hyperparameters present for our I-Forest model. We consider two of the many hyperparameters of the I-Forest, such as the number of trees in the forest and the contamination rate. After performing a cross-validation, we keep a total number of trees in the forest at 100, resulting in better performance but at the cost of increased computation time compared to smaller values. We maintain a contamination rate as well of 10%, as described originally in [66] which led to our performance seen in the results.

For our neural-network models such as the RBM and GAN, we also adapted several hyperparameters to improve the performance of our models. There are multiple hyperparameters present for the RBM. Our hyperparameters considered for tuning the RBM model include the number of hidden units. The amount of hidden units in the RBM directly influences the complexity of the RBM. With a larger amount of hidden units, a greater representation of the data can be obtained in return for an increase in computational time. We perform cross validation on this hyperparmeter and maintain 256 as the optimal amount of hidden units for both the RBM and DBN. For our DBN, we opt to implement two hidden layers; that is, there are two sequential RBMs utilized for our DBN model. As demonstrated in our experimental results, our RBM, especially with respect to the newer DNP3 Intrusion Detection dataset, had the best results. Our GAN-based model additionally has several hyperparameters as described in [4]. We maintain most hyperparameters as described in the original work. Notably, as opposed to two-dimensional images, we instead are supplying network traffic which is one-dimensional. As mentioned above, our testing process involves the calculation of the threshold value of the scaled anomaly scores. Using the scaled anomaly scores, we then determine an optimal threshold value that maximizes our metric of interest. From our experimental results, the best threshold value was kept at a 10%. This threshold value of anomalies effectively maximizes both accuracy and our specific metric, G-Mean. Table 2.2 provides a summary of our hyperparameters applied in our models.

Table 2.2: Summary of key hyperparameters in our unsupervised learning models.

Model	Key Hyperparameters	Details
OCSVM	Kernel Gamma	Radial Basis Function Gamma = 0.001
I-Forest	Number of Trees Contamination Rate	100 Trees 10% Contamination Rate
RBM	Number of hidden units	256 hidden units
DBN	Number of hidden units Number of hidden layers	256 hidden units 2 hidden layers
GAN	Hyperparameters Anomaly scores threshold	Follows settings from [4] 10% threshold value

2.4 Numerical Results

In this section, we first present the results of the five experiments described in Section 2.3.3, namely, applying the OCSVM, I-Forest, RBM, DBN, and GAN-based anomaly detection models across our four datasets. We then particularly examine the performance the GAN-based model against other models. This is to draw further insights as to the best of our knowledge, this model has not been previously used in conjunction of our datasets.

2.4.1 Experiment W-I

Table 2.3 presents the results of Experiment W-I. We observe that for this initial experiment, the best performing model with respect to the G-Mean and additionally having the best 1-FPR was the I-Forest for the newer NSL-KDD dataset, the best performing model for the KDD-Cup 99 was the DBN which statistically outperformed all other models in all performance metrics, and the best performing model for the CIC-IDS2017 dataset was the RBM, which also statistically outperformed all other models in all performance metrics. With respect to the NSL-KDD dataset, the I-Forest's results were not statistically significant save for the 1-FPR compared with the OCSVM and the RBM models. The GAN-based method was not as effective at distinguishing and classifying anomalies in network traffic on the NSL-KDD or KDD-Cup 99 datasets.

Table 2.3: Experiment W-I: The mean and standard deviation for the NSL-KDD, KDD-Cup 99, and CIC-IDS2017 Datasets. Statistical significance at 5% is indicated by an asterisk (*).

	NSL-KDD					KDD-Cup 99					CIC-IDS2017				
	OCSVM	I-Forest	RBM	DBN	GAN	OCSVM	I-Forest	RBM	DBN	GAN	OCSVM	I-Forest	RBM	DBN	GAN
Accuracy	95.69	**96.98**	96.50	88.63	59.32	95.47	96.64	98.84	**99.97***	91.64	50.42	83.41	**94.80***	60.27	62.57
	(2.55)	**(0.55)**	(0.45)	(0.26)	(3.49)	(0.27)	(0.17)	(0.12)	**(0.02)**	(7.75)	(0.11)	(2.63)	**(0.19)**	(3.28)	(8.26)
Precision	98.59	**99.95**	96.54	89.92	87.98	97.12	**99.97***	98.84	**99.97***	76.61	20.36	60.25	**94.12***	43.91	64.51
	(1.27)	**(0.02)**	(0.44)	(0.21)	(10.36)	(0.26)	**(0.04)**	(0.12)	**(0.02)**	(17.57)	(0.08)	(7.92)	**(0.56)**	(3.67)	(14.37)
Recall	96.57	96.81	99.76	**100***	41.77	98.20	96.53	99.95	**99.99***	98.73	52.16	47.30	**78.31***	72.55	54.59
	(3.02)	(0.50)	(0.07)	**(0.00)**	(19.49)	(0.44)	(0.18)	(0.03)	**(0.01)**	(0.87)	(0.12)	(8.40)	**(1.38)**	(3.30)	(16.85)
1-FPR	95.91	**99.45***	93.36	89.92	98.80	92.81	99.45	99.73	**99.97***	97.67	49.98	92.25	**98.82***	23.12	54.32
	(2.51)	**(0.30)**	(0.89)	(0.11)	(1.03)	(0.00)	(1.01)	(0.22)	**(0.00)**	(1.76)	(0.09)	(2.51)	**(0.38)**	(0.44)	(27.26)
F1-Score	97.51	**98.36**	96.25	83.30	50.00	97.66	98.22	99.84	**99.97***	84.12	29.28	52.74	**85.56***	47.14	36.03
	(1.54)	**(0.25)**	(0.50)	(0.38)	(10.82)	(0.15)	(0.09)	(0.13)	**(0.02)**	(12.92)	(0.11)	(7.70)	**(0.66)**	(0.29)	(2.98)
G-Mean	96.20	**98.12**	96.05	94.83	56.70	95.47	97.98	99.81	**99.98***	85.82	51.07	65.84	**87.97***	40.94	50.63
	(0.02)	**(0.29)**	(0.00)	(0.11)	(8.38)	(0.00)	(0.49)	(0.00)	**(0.01)**	(11.08)	(0.11)	(6.61)	**(0.72)**	(0.38)	(8.63)

2.4.2 Experiment W-II

Table 2.4 presents the results of Experiment W-II. As seen in the table, when using a six-feature subset corresponding to features present in an SDN environment, we find that the DBN is more effective on all of our intrusion detection datasets at classifying anomalies by obtaining the statistically highest G-Mean out of all the models. The OCSVM is statistically better than the I-Forest with respect to accuracy, while the RBM is better with respect to recall on the NSL-KDD dataset; The DBN model had the highest recall out of all the metrics. In comparison, on the KDD-Cup 99 dataset, the GAN outperformed all models except the DBN on accuracy, and the I-Forest outperformed our other models on precision. The OCSVM did have a statistically better performance on the F1-score metric as well. Observed is that the RBM tended to perform the best on most metrics; the DBN was the best model statistically with respect to the G-mean. Precision was best in the OCSVM and I-Forest, while the RBM, DBN, and GAN performed the best with respect to recall.

2.4.3 Experiment W-III

Table 2.5 presents the results of Experiment W-III. The DBN model statistically outperformed all other models in terms of the geometric mean. The DBN also had the highest recall score of all the models. The I-Forest exhibited the best performance with precision on the NSL-KDD and KDD-Cup 99 datasets, while the DBN had the best precision on the DNP3 intrusion detection dataset. The I-Forest also had the best accuracy on the DNP3 dataset. The OCSVM has statistically better results on the accuracy for the NSL-KDD dataset and had the highest 1-FPR. On the KDD-Cup 99 dataset, the I-Forest is the best model with respect to the precision and 1-FPR. The novel RBM-GAN approach proves to have better performance for accuracy and recall, outperforming the OCSVM and I-Forest. The performance of the DBN was the best statistically with respect to the G-Mean, along with performing well on the precision and recall metrics.

Table 2.4: Experiment W-II: The mean and standard deviation for the NSL-KDD and KDD-Cup 99 dataset using a subset of features present in SDN traffic alongside the DNP3 dataset. Statistical significance at 5% is indicated by an asterisk (*).

	NSL-KDD					KDD-Cup 99					DNP3				
	OCSVM	I-Forest	RBM	DBN	GAN	OCSVM	I-Forest	RBM	DBN	GAN	OCSVM	I-Forest	RBM	DBN	GAN
Accuracy	**95.16***	89.04	91.91	89.98	51.49	95.08	86.17	96.68	96.19	**97.44**	55.4	84.92	**91.05**	90.91	90.42
	(1.45)	(1.15)	(2.01)	(0.61)	(3.70)	(0.13)	(0.93)	(0.82)	(0.11)	**(1.50)**	(3.51)	(0.62)	**(0.17)**	(0.04)	(0.68)
Precision	97.35	**98.43**	92.63	90.16	72.83	97.43	**99.11**	95.33	96.33	91.00	**100**	**100**	91.08	91.74	90.87
	(1.35)	**(1.81)**	(1.68)	(0.48)	(5.94)	(0.06)	**(1.02)**	(1.73)	(0.11)	(6.66)	**(0.00)**	**(0.00)**	(0.17)	(0.03)	(0.07)
Recall	97.33	92.91	99.98	**100**	22.31	97.46	85.79	99.01	**100**	95.54	55.4	84.99	**100**	**100**	99.45
	(0.86)	(0.89)	(0.02)	**(0.00)**	(6.33)	(0.15)	(1.00)	(1.15)	**(0.00)**	(0.60)	(3.52)	(0.68)	**(0.00)**	**(0.00)**	(0.74)
1-FPR	89.46	**99.63***	84.56	90.20	73.84	92.74	**99.84***	94.45	98.15	91.00	55.40	85.00	**99.01***	95.78	90.87
	(5.68)	**(0.44)**	(3.71)	(0.63)	(5.94)	(0.01)	**(0.18)**	(2.30)	(0.05)	(6.65)	(3.51)	(0.68)	**(0.00)**	(0.05)	(0.07)
F1-Score	92.32	**92.91**	89.03	83.79	33.73	**97.44***	91.61	95.06	94.32	92.96	71.19	91.89	**95.31***	86.58	94.96
	(0.80)	**(0.89)**	(3.50)	(0.88)	(7.39)	**(0.07)**	(0.88)	(1.22)	(0.17)	(3.66)	(2.84)	(0.40)	**(0.09)**	(0.05)	(0.38)
G-Mean	93.24	93.94	91.91	**94.97***	40.16	95.07	92.54	96.68	**98.15***	93.11	54.80	85.00	91.04	**95.78***	95.06
	(3.42)	(0.01)	(0.02)	**(0.25)**	(6.57)	(0.00)	(0.01)	(0.01)	**(0.01)**	(3.44)	(3.51)	(0.68)	(0.16)	**(0.05)**	(0.39)

Table 2.5: Experiment W-III: The mean and standard deviation for the NSL-KDD, KDD-Cup 99, and DNP3 Datasets models paired with the Restricted Boltzmann Machine. Statistical significance at 5% is indicated by an asterisk (*).

	NSL-KDD				KDD-Cup 99				DNP3			
	RBM+ OCSVM	RBM+ I-Forest	RBM+ DBN	RBM+ GAN	RBM+ OCSVM	RBM+ I-Forest	RBM+ DBN	RBM+ GAN	RBM+ OCSVM	RBM+ I-Forest	RBM+ DBN	RBM+ GAN
Accuracy	**91.03***	84.51	88.68	64.79	94.77	88.92	96.29	**97.54***	90.93	**91.14**	90.91	87.17
	(2.99)	(2.13)	(0.66)	(6.19)	(0.69)	(0.27)	(0.07)	**(1.72)**	(0.16)	**(0.14)**	(0.04)	(3.19)
Precision	93.96	**99.71***	89.96	68.59	97.30	**100.0**	96.43	90.95	90.92	91.14	**91.74**	90.70
	(2.66)	**(0.60)**	(0.51)	(7.67)	(0.23)	**(0.00)**	(0.07)	(7.73)	(0.17)	(0.14)	**(0.03)**	(0.23)
Recall	98.03	89.65	**100***	79.51	94.77	89.26	**100***	96.91*	**100**	**100**	**100**	96.91
	(0.20)	(2.31)	**(0.00)**	(9.09)	(0.69)	(0.88)	**(0.00)**	(1.61)	**(0.00)**	**(0.00)**	**(0.00)**	(1.61)
1-FPR	**99.01**	**99.01**	89.96	90.70	93.75	**99.44***	96.20	90.95	**99.01**	**99.01**	91.74	90.69
	(0.00)	**(0.00)**	(0.72)	(0.02)	(0.01)	**(0.09)**	(0.01)	(0.08)	**(0.00)**	**(0.00)**	(0.03)	(0.23)
F1-Score	88.44	**91.72**	83.36	72.15	94.15	92.79	**94.46**	93.45	95.25	**95.37**	86.58	93.39
	(4.50)	**(3.40)**	(0.94)	(2.89)	(0.31)	(0.74)	**(0.10)**	(4.17)	(0.09)	**(0.08)**	(0.05)	(1.88)
G-Mean	93.24	93.40	**94.85***	73.08	96.07	94.30	**98.20***	93.69	90.92	91.11	**95.71***	93.46
	(2.27)	(1.62)	**(0.27)**	(2.70)	(0.12)	(0.14)	**(0.03)**	(3.89)	(0.16)	(0.14)	**(0.02)**	(1.90)

2.4.4 Experiment A-I

Table 2.6 presents the results of Experiment A-I. As seen in the table, when training and testing *across* datasets to evaluate the generalizability of the models, we observed some differences as compared to *within* datasets (as in Experiments W-I–III). For the mixed datasets, the RBM and I-Forest generally outperformed other models. However, interestingly, neither one consistently outperformed the other. That is, the RBM performed best when training on NSL-KDD and testing on KDD-Cup 99, and I-Forest vice versa. We did observe the best performance with respect to recall on the DBN dataset. The GAN-based model, while still being outperformed, had performance similar to that of the other models when tested using the KDD-Cup 99 dataset. Unfortunately, the GAN-based model did not perform well when tested on the NSL-KDD dataset. The drop in performance across models is potentially indicative of the underlying datasets; the NSL-KDD dataset has removed redundant observations from the KDD-Cup 99 dataset and has significantly addressed major class imbalances in the dataset [114]. As a result, however, training on the older KDD-Cup 99 dataset may lead to reduced overall performance.

2.4.5 Experiment A-II

Table 2.7 presents the results of Experiment A-II. Recall that, as discussed in section 2.3.3, this experiment consisted of applying the same across dataset training and testing, instead now applying the RBM for feature engineering for our analysis. The respective model is trained on the dataset present in the larger header of the table. We find that when applying the RBM, the OCSVM has superior performance compared with the other models when trained on the NSL-KDD and tested on the KDD-Cup 99 dataset on both the F1-score and G-mean metrics. We also find that the the GAN outperforms both OCSVM and I-Forest with respect to accuracy when tested on the KDD-Cup 99 dataset. For testing procedures on the NSL-KDD dataset, the I-Forest statistically outperformed all models with respect to the G-Mean, precision,

Table 2.6: Experiment A-I: The mean and standard deviation for training and testing across different datasets to determine generalizability of our models. Statistical significance at 5% is indicated by an asterisk (*).

	NSL-KDD Train, KDD-Cup 99 Test					KDD-Cup 99 Train, NSL-KDD Test				
	OCSVM	I-Forest	RBM	DBN	GAN	OCSVM	I-Forest	RBM	DBN	GAN
Accuracy	97.80 (0.09)	97.91 (0.07)	**99.97 (0.01)**	96.23 (0.13)	97.68 (2.37)	95.48 (0.05)	96.62 (0.20)	**97.22 (0.82)**	86.42 (4.64)	74.54 (1.99)
Precision	99.20 (0.06)	99.95 (0.01)	**99.98 (0.01)**	96.37 (0.11)	92.81 (0.01)	97.67 (0.04)	**99.92* (0.03)**	97.25 (0.80)	88.44 (3.10)	93.93 (1.99)
Recall	98.51 (0.04)	97.84 (0.07)	99.53 (0.26)	**100* (0.00)**	98.72 (0.38)	97.22 (0.03)	96.26 (0.23)	99.67 (0.13)	**100* (0.00)**	59.22 (4.16)
1-FPR	97.10 (0.14)	99.79 (0.17)	**99.99* (0.00)**	96.39 (0.12)	92.81 (8.32)	93.77 (0.08)	**99.41* (0.22)**	80.56 (7.30)	88.42 (0.03)	93.93 (1.99)
F1-Score	98.85 (0.05)	98.90 (0.03)	99.83 (0.09)	**98.90 (0.03)**	94.38 (0.19)	97.44 (0.04)	**98.06 (0.12)**	97.05 (0.95)	80.18 (6.52)	72.44 (2.91)
G-Mean	97.80 (0.09)	98.81 (0.08)	**99.76 (0.13)**	98.17 (0.06)	95.52 (4.63)	95.48 (0.05)	**97.83* (0.14)**	88.89 (4.02)	94.03 (1.67)	74.48 (2.40)

Table 2.7: Experiment A-II: The mean and standard deviation for training and testing across different datasets to determine generalizability of our models when combined with the RBM. Statistical significance at 5% is indicated by an asterisk (*).

	NSL-KDD Train, KDD-Cup 99 Test				KDD-Cup 99 Train, NSL-KDD Test			
	RBM+ OCSVM	RBM+ I-Forest	RBM+ DBN	RBM+ GAN	RBM+ OCSVM	RBM+ I-Forest	RBM+ DBN	RBM+ GAN
Accuracy	95.93 (0.18)	88.84 (0.36)	96.36 (0.20)	**98.44* (0.60)**	**89.22* (0.03)**	83.07 (0.53)	85.91 (3.55)	74.59 (2.50)
Precision	95.34 (0.21)	**99.99* (0.00)**	96.50 (0.18)	93.75 (3.37)	86.62 (0.06)	**99.87* (0.14)**	87.99 (2.43)	93.66 (1.79)
Recall	98.47 (0.01)	92.29 (0.39)	**100* (0.00)**	97.59 (1.17)	99.14 (0.04)	93.70 (0.59)	**100* (0.00)**	59.59 (5.68)
1-FPR	95.34 (0.21)	96.36 (0.08)	**96.49 (0.18)**	93.75 (3.37)	86.82 (0.07)	**99.87* (0.14)**	87.99 (2.43)	94.66 (1.79)
F1-Score	**95.59 (0.19)**	92.40 (0.19)	94.58 (0.29)	95.56 (1.64)	85.80 (0.10)	85.71 (0.29)	**88.84* (0.36)**	72.46 (3.88)
G-Mean	96.89 (0.11)	94.26 (0.19)	**98.23* (0.09)**	95.62 (1.60)	92.78 (0.03)	**96.74* (0.32)**	93.80 (1.31)	74.50 (3.04)

and 1-FPR, however, the OCSVM was statistically superior on accuracy. The DBN was statistically superior on the recall metrics for both train and test scenarios. Our GAN was unfortunately not as generalizable as our other models when trained and tested across datasets.

2.4.6 GANomaly Performance

While in many cases in literature, novel GAN-based models seem to perform better than other ML models for anomaly detection [65, 130, 19], our results generally show that our proposed GAN from [4] was only able to perform better on certain models in certain instances on our first two datasets. Table 2.8 summarizes instances in which the GAN-based model outperformed other ML models in the study.

As described in the table, when applied to the NSL-KDD cybersecurity dataset, the GAN-based method performed better when paired with the RBM and using the KDD-Cup 99 as the testing dataset based on G-Mean. Interestingly, in general, the GAN performed better than the I-Forest on most metrics when paired with the RBM and using the KDD-Cup 99 test dataset. The GAN-based model tended to outperform other ML models on the KDD-Cup 99 dataset on other metrics. When using the KDD-Cup 99 dataset, our GAN-based model outperformed all four other models on accuracy in an SDN context, but it was only able to outperform the I-Forest on G-Mean in that instance. The GAN did perform better than the OCSVM, RBM, and DBN in some instances. Our GAN-model tended to perform better when paired with the RBM with the KDD-Cup 99 dataset compared with other models with other datasets and when paired in conjunction with the RBM. This observation suggests that our GAN-based anomaly detection method, while not outperforming all models in traditional networking, might be of benefit in certain scenarios for traditional networking traffic.

Table 2.8: Summary of performance comparison between GAN and other models presented in the study, where GAN outperformed other models in certain metrics (in bold). Asterisks indicate statistically significant results at an alpha of 5%. The GAN-based model was unable to outperform the other models when trained and tested on the instances of full, SDN-feature, and RBM with feature extraction on the NSL-KDD dataset.

	NSL-KDD			KDD-Cup 99				DNP3		CIC-IDS2017
	Full Features	KDD-Cup 99 Test Dataset	RBM+ KDD-Cup 99 Test Dataset	Full Features	SDN Features	RBM+ Model	RBM+ NSL-KDD Test Dataset	SDN Features	RBM+ Model	Full Features
Accuracy	-	-	**OCSVM, I-Forest***	-	**OCSVM, I-Forest, RBM, DBN**	**OCSVM, I-Forest*, DBN**	-	**OCSVM, I-Forest***	-	**OCSVM*, DBN**
Precision	-	-	-	-	-	-	**OCSVM, DBN**	-	-	**OCSVM*, I-Forest, DBN***
Recall	-	**OCSVM, I-Forest**	**I-Forest**	**OCSVM, I-Forest**	-	**OCSVM, I-Forest**	-	**OCSVM, I-Forest***	-	**OCSVM, I-Forest**
1-FPR	**OCSVM, RBM DBN**	-	-	**OCSVM**	-	-	**OCSVM, DBN**	**OCSVM, I-Forest*,**	-	**OCSVM*, DBN***
F1-Score		-	**I-Forest, DBN**	-	**I-Forest**	**I-Forest**	-	**OCSVM, I-Forest* DBN**	**DBN***	**OCSVM***
G-Mean		-	**I-Forest**	-	**I-Forest**	-	-	**OCSVM, I-Forest, RBM***	**OCSVM, I-Forest, RBM***	**DBN***

The performance of the GAN was significantly better than when trained and tested on our SDN-SCADA/ICS dataset. From experimental results, our GAN outperformed the other models within this SDN environment, even performing statistically better compared versus the OCSVM and I-Forest in several metrics. Our GAN performance on these datasets presumably is indicative of the training process; our GAN-based method is trained without attack traffic, only on benign traffic. The generator constructs anomalous samples for the discriminator to detect, but does not replicate attack traffic exactly. While this process might have reduced the performance of the GAN on the NSL-KDD and KDD-Cup 99 datasets, it reflected improved the performance when examined on the DNP3 dataset.

Of special note was the performance of our GAN-based anomaly detection model on the CIC-IDS2017 dataset. While most of our models struggled with classifying anomalies on the dataset, our GAN demonstrated higher performance compared to other benchmark models. We observe that our GAN-based method tends to statistically outperform the OCSVM in all metrics save the G-Mean; our GAN statistically outperformed the DBN on the precision, 1-FPR, and the G-Mean metrics and outperformed the I-Forest on the recall and precision metrics. Our GAN's performance demonstrates that it can handle anomaly detection better than the OCSVM and DBN when faced with more classes of anomalies such as the multitude of anomalies present in the CIC-IDS2017 dataset.

2.5 Conclusion and Future Works

Effective cybersecurity depends upon the proper detection of and response to anomalies encountered on network traffic. Ensuring the security of information systems highly depends on the defensive countermeasures employed. These defenses include the usage of network-based intrusion detection and prevention systems. This study provides a robust approach to detecting anomalous network traffic stemming from various threat actors in cyberspace.

In particular, this study introduces a GAN-based unsupervised/semi-supervised approach that trains only on non-anomalous samples of network data, in addition to presenting other unsupervised methods, including the OCSVM, I-Forest, RBM, and DBN for analysis. The GAN-based approach consists of a generator and a discriminator network trained in an adversarial manner. The generator learns the distribution of non-anomalous data in the training process, which is consequently used to reconstruct input data. This trained generator is then used to detect anomalies. When an anomalous sample is introduced, the trained generator reconstructs the sample so that the sample loses anomalies. As the sample created by the generator does not have any anomalies, there are differences between the input sample and the corresponding sample that was reconstructed, which allows the GAN to detect anomalies.

We test this hypothesis and demonstrate the application of the GAN-based model in addition to the OCSVM, I-Forest, RBM, and DBN approach introduced through various experiments with different models. Our results show that the GAN-based anomaly detection approach generally has performance either equal to or slightly worse than other models; however, our RBM-based and DBN-based methods either performed as well or better than other models. Generally, from our observations, the I-Forest was superior when tested on the NSL-KDD dataset and the DBN tended to perform best on the KDD-Cup 99 and DNP3 dataset. Additionally, the GAN's performance consistently did improve when we added an RBM for feature extraction, which is in contrast to our other models which had similar performance with and without the RBM for feature extraction on the NSL-KDD and KDD-Cup 99 datasets. Our RBM performed the best across all metrics on the newer CIC-IDS2017 dataset; while our GAN did not have the best performance, it managed to outperform other benchmark models on CIC-IDS2017 dataset.

Since unsupervised models do not need labeled data, akin to malware signatures, these unsupervised models could be beneficial in detection of zero-day attacks. Fortunately, all models proposed could potentially contribute to creating effective intrusion detection and prevention systems for cybersecurity engineering through

the identification of network anomalies. Anomaly-based intrusion detection should be supplemented with other mitigation mechanisms such as rule-based firewalls and access control lists in order to better provide for a greater defense-in-depth approach to organizational cybersecurity. Our models, when combined with these additional mechanisms, can provide greater network security.

There are several research directions for future work. First, we can opt to use actual packet capture data which captures a more holistic representation of network traffic passing between devices and information systems. In addition, this approach allows for further study of the generalizability of the models to novel network attacks, including zero-day exploits. Second, while this work analyzed primarily traditional and software-based networking environments, there could be different approaches that may be suitable for other networking environments. Third, although the data used in this study are time series data, this approach can also be tested instead on malware samples as opposed to network traffic which can potentially lead to improvements in classifying cybersecurity threats.

Chapter 3

FLOWS: Federated Learning Optimal Weighting of Sensors for Network Intrusion Detection Systems

The recent advancements in computing and information technology have allowed for society to become more interconnected. However, as an inherent consequence of this rapid growth, more cybersecurity vulnerabilities are present in everyday computing and information systems. As described by the National Institute of Standards and Technology (NIST), the number of vulnerabilities present in connected devices has continually increased over time, with the rate of known high and critical vulnerabilities increasing substantially during the past ten years [83]. This issue thus presents a critical need to defend these systems against cyber threats.

Network Intrusion Detection Systems (NIDS) present an effective method to defend against cybersecurity threats [23]. As opposed to other security mechanisms such as firewalls and access control lists (ACLs), NIDS often have the unique advantage of being intelligent; they are often equipped with an algorithm to discern

normal traffic from anomalous or malicious traffic. Firewalls and ACLs, however, generally only follow a predefined set of rules as to whether to permit traffic or deny traffic. Henceforth, organizations incorporating a NIDS into their security posture gain the added benefit of increasing their overall security through an additional layer of defense which is able to detect and prevent threats from causing major organizational harm.

In addition to the benefits offered by NIDS, federated learning, as proposed initially in [72], offers added advantages for data-centric organizations. The hallmark of federated learning is the ability for clients to be able to train on their local data without having to send data directly to a centralized server. Instead, local model weights from edge clients are sent to a centralized server. The centralized server then averages the model weights from the edge clients, which then are sent back to the edge clients. This process repeats in an iterative manner until the model is trained. A direct benefit of federated learning is maintaining the privacy of data. As opposed to having to send data to a centralized server for training and inference, the training is performed on the edge; model weights are instead sent to the centralized server. This allows organizations to more readily adhere to important data protection regulations and frameworks, such as the General Data Privacy Regulations (GDPR) [47], the Health Insurance Portability and Accountability Act (HIPAA) [12], and the Payment Card Industry Data Security Standards (PCI-DSS) [78], among many others.

In addition to adherence to the data privacy and security frameworks, federated learning can maintain data privacy within organizations. Good networking practices suggest to divide larger networks into smaller, more manageable subnets [90]. Organizations deploying NIDS for individual subnets can leverage federated learning in order maintain the data privacy of each subnet.

A major limitation to the benefits of federated learning, however, is the inability of the federated learning averaging algorithm to consider the underlying performance and health of the edge clients. This work proposes a novel framework to provide client weights by taking into consideration the underlying client state in order to

best contribute to a federated average for a centralized, federated learning system. We thus present FLOWS, **F**ederated **L**earning **O**ptimal **W**eighting of **S**ensors. We present our FLOWS framework as a partially observable Markov decision process (POMDP) in which the underlying state of the clients are unknown. We implement a recurrent Q-learning reinforcement learning (RL) algorithm in order to solve our POMDP. We find that in conjunction with our reinforcement learning algorithm, our novel federated learning framework improves overall classification performance and generally demonstrates a better accuracy, F1-score, and G-Mean compared to traditional federated learning.

3.1 Literature Review

Because our work covers multiple fields of study, we divide our literature review into three primary sections: federated learning for network intrusion detection, optimal client selection and weighting, and prescriptive models for classification tasks. We provide a tabular overview of the surveyed literature in Table 3.1.

Table 3.1: An overview of related literature and attributes of each work sorted chronologically.

	Attributes			
Reference	Intrusion Detection	Federated Learning	Weighting	Prescriptive
[132]	X	X		
[21]	X	X		
[33]	X			X
[131]		X	X	
[125]		X	X	
[89]		X	X	
[129]		X	X	
[122]		X	X	
[67]				X
[121]				X
[27]	X			X
[6]	X	X		
[50]	X	X		
[97]	X	X		X
Our Work	X	X	X	X

3.1.1 Federated Learning for Network Intrusion Detection

Intrusion detection serves a crucial role in the defense of many networks [23], [90], [11]. When coupled with other existing defenses such as firewalls and ACLs, intrusion detection systems can better defend networks from cyber attacks. Most effective intrusion detection systems rely on the application of an algorithm for effective detection of threats [23], [11]. Respectively, the introduction of federated learning has several benefits for the field of intelligent intrusion detection; namely, federated intrusion detection provides a privacy preserving method for training of intrusion detection systems. In the work presented in [6], several federated machine learning models were applied toward internet of medical things (IoMT) utilizing popular intrusion detection datasets. Unsupervised federated learning was also implemented in [50]; an autoencoder-based federated learning solution outperformed other unsupervised models such as Generative Adversarial Networks for distributed intrusion detection.

In [21], the authors applied a federated learning framework incorporating gated recurrent units (GRUs) for wireless intrusion detection. A new framework developed in [21] improved model accuracy by 8% compared with centralized learning schemes and also demonstrated robustness against poisoning attacks. Other recurrent network structures demonstrate improved performance when applied with federated learning. In the work of [132], long-short term memory recurrent neural networks and convolutional neural networks used in conjunction with federated learning provided results which were as effective as centralized learning.

3.1.2 Optimal Client Selection and Weighting

As originally proposed in the work of [72], federated learning is a novel paradigm that intends to respect the privacy of the clients while still allowing for a global model to be updated on a centralized server via client weights. The prevailing algorithm used, FedAvg, still has some shortcomings;specifically, edge client performance is not

considered for federated contributions. Several works have intended to improve upon the FedAvg algorithm to improve global model performance.

In the work presented in [89], a modified FedAvg algorithm was implemented in the presence of Byzantine clients; Byzantine clients are clients which potentially send arbitrary or malicious data. They proposed the usage of a weight-truncation-based pre-processing technique and found that their technique was comparable without attacks and maintained a robust performance in the presence of malicious clients. Another modified FedAvg algorithm was implemented in the work of [131]. Notably, the authors made the distinction wherein clients either possess identically independently distributed (IID) data or non-IID data; federated learning oftentimes incorporate clients that possess non-IID data. Their approach, CSFedAvg, leverages clients with IID data instead to contribute to the global model more than non-IID clients which alleviates an issue presented originally in [72]. Similarly, in the work presented in [129], a client selection scheme was implemented for federated learning; a noteworthy aspect is the presence of noise among clients which affects not just the feature set but the target/label set for those clients. Their experimental results indicate that their proposed client selection technique performed as well or outperformed other baseline models.

As implemented in [122], edge clients' contributions to the global average was determined by a probabilistic model based upon the edge client's underlying data distribution; clients had a higher probability of being selected to contribute to the global mean if they had IID data as opposed to clients with non-IID data. Additionally, similar work was conducted in the work of [125]. As proposed by the work in [125], the researchers applied a client weighting strategy by instead measuring the resources expended per client. Their algorithm, after experimental analysis, was able to outperform the baseline model by more efficiently allocating resources to edge clients in a manner that maintained long term performance instead of short-term, rapidly changing client configurations. While these works are promising for client selection and weighting, a critical discrepancy of these works is that they

do not directly provide a prescriptive method based upon underlying edge clients' health or performance; all methods maintained full observability of their federated environments which could provide challenging or impossible in certain environments.

3.1.3 Prescriptive Methods for Classification Tasks

While the aforementioned works provide several benefits in improving the federated learning averaging algorithm, notably, there was full knowledge of edge client states. This notion might not be as realistic in scenarios in which the underlying client health or client state is unable to be fully observed which is an issue in many organizational networks. Other methods need to be applied in order to address this lack of full observability in federated learning systems for cybersecurity. Works previously implementing partial observability via a prescription-based method for classification include [67] which saw benefits in developing a two-stage approach for sepsis prediction in hospitalized patients. Additionally, [121] implemented a similar two-stage approach for anomalous sensor traffic in autonomous vehicles and found that the two-stage approach improved detection of anomalies in vehicular traffic. Prescriptive approaches for the online video gaming industry were applied in previously in [33]; their model had promising results in prompt detection of anomalies. Further prescriptive works like in [97] considered a two-stage approach but did not consider the weighting of edge clients.

While a prescriptive, two-stage approach has several benefits, we find a lack of works in the present literature addressing such a two-stage approach in the literature. Our work specifically aims to further the field of cyber threat identification and network intrusion detection in federated learning, federated client selection and weighting, and prescriptive classification. Consequentially, our work aims to address this gap in the intrusion detection literature through implementing the following:

- We formulate the federated intrusion detection problem instead using a two-stage framework. This framework incorporates federated learning in the first

stage, and a POMDP in the second stage. The federated learning first stage sends observations to be applied in the POMDP second stage.

- Our second stage POMDP is solved using a deep reinforcement learning algorithm, namely, the Deep Recurrent Q-Network (DRQN). Our DRQN maintains a beliefs of the edge clients' underlying state and outputs a predicted action corresponding to the classification of the underlying federated learning system. These actions then correspond to weights for each client's contribution to the federated average.

- We compare our prescriptive, two-stage approach with the traditional federated learning algorithm. Our results demonstrate that our two-stage framework directly improves performance and achieves the best recall; more anomalies are classified correctly using our approach.

Our choice to frame the problem as a POMDP is directly motivated by the fact that the true health of each edge client is not fully observable in some server-client relationships. Henceforth, our work provides not just a realistic solution, but a direct improvement in the global model's performance. We describe our FLOWS framework graphically in Figure 3.1.

3.2 Methodology

3.2.1 NSL-KDD Dataset

In order to simulate network traffic with attacks, we utilize the popular NSL-KDD intrusion detection dataset as originally proposed in [114]. The NSL-KDD dataset is an improvement upon the original KDD-Cup 99 dataset that was originally proposed by the Defense Advanced Research Projects Agency (DARPA) and MIT Lincoln Labs [61]. Compared to the previous KDD-Cup 99 dataset, the NSL-KDD dataset has removed redundant and duplicate records; the distribution of attacks have also

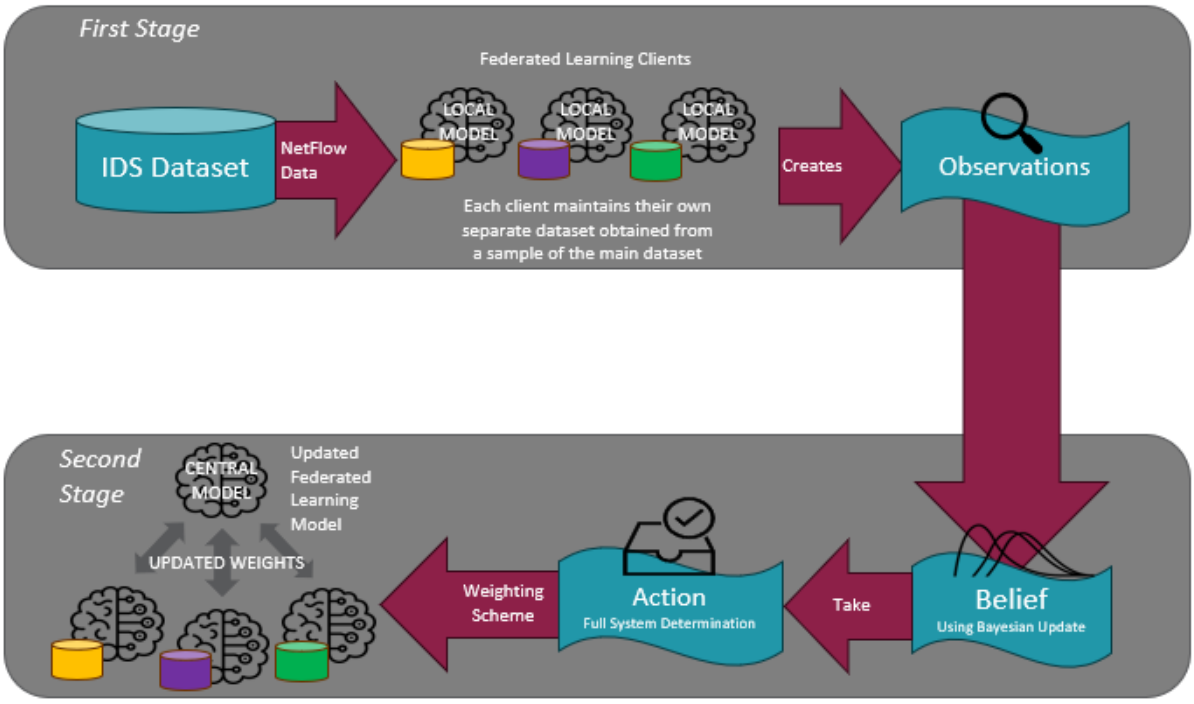


Figure 3.1: Our novel FLOWS framework which comprises of two primary stages. Stage one consists of obtaining observations from our first stage of our federated learning model. These observations are then passed to the second stage of our model which are then used to update the belief of the edge clients and the full system. Our RL agent then takes an action which corresponds to a weighting schema for normal and attacked nodes in the system. These new weights are then passed back to an updated federated averaging algorithm.

been adjusted to be more proportional and representative for network traffic. The improved NSL-KDD dataset's records are meant to represent network traffic from a local area network. The dataset comprises of classes of attack traffic:

- DoS: Denial of Service attacks; these attacks focus on overwhelming a target in order to take the system offline.
- Probe: Probing attacks often occur as part of the reconnaissance phase of cyber attacks, as described in the MITRE ATT&K chain [124]. These attacks are synonymous with scanning.
- U2R: User-to-Root, these attacks happen when a user is able to escalate privileges to gain greater control over a victim system.
- R2L: Remote-to-Local, these attacks take over a local machine on an internal network by an unauthorized, external attack source.

The dataset is balanced with respect to the amount of normal traffic and attack traffic; however, there is an imbalance of attack traffic among the four different types of attacks. A representation of the distribution is depicted with the four attack classes alongside normal traffic is depicted in Figure 3.2.

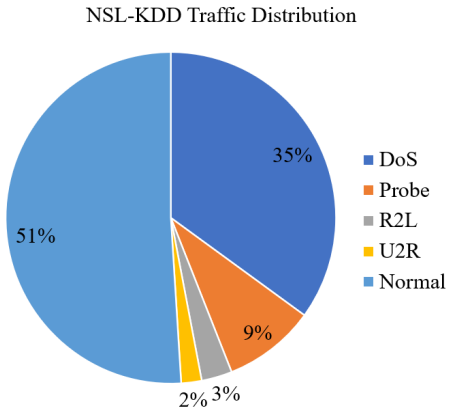


Figure 3.2: Distribution of the traffic present in the NSL-KDD dataset.

For both our federated learning benchmark and novel FLOWS framework, we divide our training dataset into multiple sub-datasets corresponding to the number of total federated clients. The output of the first stage, the federated learning client performance, is taken as observations for the second stage. After the weight multipliers are assigned to the edge clients, we apply the testing dataset to both the benchmark federated learning and our FLOWS framework.

3.2.2 Federated Learning

As discussed previously, federated learning enables individual clients to independently train on their own data, thus maintaining privacy of data across clients. Weights are passed from client to centralized server in order to update a shared, global model. The nominal equation for FedAvg is represented as

$$\xi_t^g = \frac{1}{N_c} \sum_{1}^{N} \xi_{t-1,i}^l \tag{3.1}$$

where ξ_t^g is the global model created at time t, N_c is the total number of clients, and $\xi_{t-1,i}^l$ is the local model received from all clients [72]. While useful for initially implementing a means to update clients, as stated, it fails to acknowledge lower-performing clients. As a result, we then modify the FedAvg to as follows:

$$\xi_t^g = \frac{1}{N_c} \sum_{1}^{N} \omega_i \xi_{t-1,i}^l \tag{3.2}$$

where ω_i is a weight multiplier for client i. This additional weighting now can consider instances in which clients perform poorly, potentially as a result of an attack; clients are weighted with respect to their performance. Since we assume that we cannot directly infer the state of each client, we thus need alternative means to assess weighting per client. We provide a method to determine the weights of our clients in the following section.

3.2.3 POMDP Formulation

As mentioned previously, since we cannot directly determine the state of each client, we intend to formulate our federated learning problem as a POMDP. A POMDP is a seven tuple, $(S, A, R, P, \Omega, O, \pi)$ which comprises of an additional Ω, O, and π corresponding to an observation probability function, observation space, and a belief state. The additional Ω, O, and π must be included compared as opposed to a normal MDP because of the partially observable underlying true state of a POMDP [60]. Our POMDP transitions between states over each federated learning update round until termination as described in Figure 3.3.

After the start of each round, we transition to a new state. This transition is based upon each client's transition probability matrix (TPM). We then take an observation; in this case, our observation is the performance of the client. We then update the client's belief and store the belief in a belief vector. Next, our RL agent takes an action and obtains a reward based upon their action taken. The process repeats and we transition to the next state.

Let $s \in S = \{0, 1, ..., \lambda\}$ represent the underlying federated learning system state. States corresponding to $s = 0$ indicate that no underlying client is under attack, all clients in the normal state, while $s = \lambda$ indicates that all clients are in the attacked state. In addition, states $s = 1, ..., \lambda - 1$ represent intermediate states. Our full state space can be represented by $0, 1, ..., 2^n - 1$ where n is the number of clients. We assume independence of our clients; each client has its own state, $j \in J = \{0, 1\}$, which represents a client being in a normal state and attacked state, respectively. In

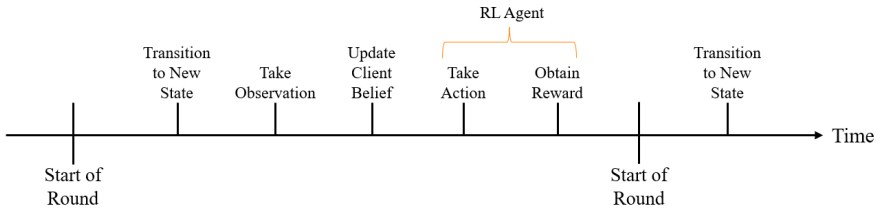


Figure 3.3: Timeline of our POMDP execution over federated learning rounds.

a two-client environment, we have four total states, $s \in S = \{0, 1, 2, 3\}$; states 1 and 2 represent instances where only client 0 is attacked and only client 1 is attacked, respectively. Note that state $3 = \lambda$ is the instance of all clients being in the attacked state.

Our POMDP actions reflect the predicted underlying state of the system; there are $a \in A = \{0, 1, ..., 2^n - 1\}$ total actions. Action $a = 0$ corresponds to marking the system as being in a not attacked state, $s = 0$, while action $a = 2^n - 1$ corresponds to all clients attacked. These actions, however, also directly correspond to the weighting of clients. In a two-client scenario, actions 1 reflects client 0 attacked and client 1 normal while action 2 reflects client 1 attacked and client 0 normal, respectively. We weight edge clients based upon the action chosen by our RL agent. In a system with n total clients with $n - z$ clients normal and z clients attacked, we can set a weight for each edge client's contribution to the federated learning average, ω_{n-z} and ω_z for weights of normal and attacked clients, respectively. We can set the weight for the normal clients, $\omega_{n-z} = \frac{1}{n-z+\frac{z}{2}}$ and for the attacked clients $\omega_z = \frac{1}{2(n-z+\frac{z}{2})}$. In a two-client scenario with one client attacked, $\omega_{n-z} = \frac{1}{2-1+\frac{1}{2}} = \frac{2}{3}$ and $\omega_z = \frac{1}{2(2-1+\frac{1}{2})} = \frac{1}{3}$. These weights are then assigned to the edge clients as shown in Equation 3.2 corresponding to the RL agent's action in our FLOWS framework.

Each edge client also transitions between states during the federated learning rounds. A new sample is obtained each round for the edge clients; we take a data-driven approach to determine the client TPMs based upon the client's historical network traffic. Respectively, each edge client i will have their own respective TPM which differs from other edge clients. This TPM is defined as:

$$P_i = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix} \tag{3.3}$$

where p is the probability of remaining in a non-attacked state and $1 - p$ is the probability of transitioning from normal to attacked. Respectively, q is the probability

of a client remaining in an attacked state and $1-q$ is the probability of transitioning from attacked to normal.

Because we cannot directly observe the state of each client, we also maintain an Ω_i which represents the probability of observing an observation o for client i given the underlying client state j. These observations correspond to the performance of each edge client on their own data. Additionally, our agent maintains a belief that the federated learning system is in a certain state. Let $\pi_t(s)$ be the full system belief of being in state s at time t, $\pi_t = [\pi_t(0), \pi_t(1), ..., \pi_t(\gamma)]$. Let $b_{i,j}(j)$ represent the belief of client i being in state j at time t. We can calculate the belief of client i being in state j' at time $t+1$ through the following equation:

$$b_{i,t+1}(j') = \frac{\Omega_i(o|j') \sum_{j \in J} P_i(j'|j) b_{i,t}(j)}{\sum_{j' \in J} \Omega_i(o|j') \sum_{j \in J} P_i(j'|j) b_{i,t}(j)} \tag{3.4}$$

where o is the current observation of client i, $j \in J = \{0, 1\}$ is the current client state of either being normal or attacked, $b_{i,t}(j)$ is the current belief of client i being in state j at time t, j' is the next state, and $b_{i,t+1}(j')$ is the belief of being in the state j' at time $t+1$. In order to calculate the full system belief, $\pi_t(s)$, we take the tensor product of the individual N client beliefs, $\pi_t(s) = b_{0,t+1}(j) \otimes b_{1,t+1}(j)...b_{N,t+1}(j)$.

Our rewards assigned to our reinforcement learning agent also correspond to the determination of the overall system state; our reward is a function of both the action taken by our RL agent given the underlying system state. Our reward function is calculated as follows:

$$R_t(s, a) = \begin{cases} R_{TN}, R_{TP} = 1, a = s \\ R_{FP}, R_{FN} = -1, a \neq s \end{cases}$$

where $R_t(s, a)$ is the reward provided to the agent by taking action a given underlying system state s. We reward our agent $R_{TN}, R_{TP} = 1$ for correct determination of the underlying system state and punish our agent for incorrect determination,

$R_{FN}, R_{FP} = -1$. Our agent has the ultimate objective of maximizing its value function. These equations extend the optimally equations proposed by [13] in order to account for a partially observable system; we can calculate value function through the following equation:

$$V(\pi_t) = \max_{a \in A} Q(\pi_t, a), a \in \{0, 1, 2, ..., 2^n - 1\} \tag{3.5}$$

Our Q-function can be calculated by the following equation:

$$Q(\pi_t, a) = \sum_{s \in S} \pi_t(s) \cdot r(s, a) + \sum_{s \in S} \pi_t(s) \sum_{o^{sys} \in O^{sys}} \Omega^{sys}(o^{sys}|s) \sum_{s' \in S} P^{sys}(s'|s) \cdot V(\pi_{t+1}) \tag{3.6}$$

where $\Omega^{sys}(o^{sys}|s)$ is the full system observation probability of observing full system observation o^{sys} given state s and $P^{sys}(s|s')$ is the full system TPM, obtained by taking a tensor product of each individual client's TPM, $P^{sys} = P_0 \otimes P_1 \otimes ... \otimes P_N$. In the terminal case, the agent obtains a reward, $V(\pi_t) = R_T(s, a)$. We provide an overview of the symbols used in our POMDP in Table 3.2.

3.2.4 Deep Recurrent Q-Network

We solve our POMDP using the popular Deep Recurrent Q-Network (DRQN). This reinforcement learning algorithm is an extension of the original Deep Q-Network originally proposed in [76]. DQNs are based upon the Q-learning algorithm commonly seen in reinforcement learning [43].

Table 3.2: An overview of both system and client symbols present in our POMDP.

System		Client	
Symbol	Representation	Symbol	Representation
s	System State	j	Client State
s'	Next System State	j'	Next Client State
O^{sys}	System Observation Space	O	Client Observation Space
o^{sys}	System Observation	o	Client Observation
Ω^{sys}	System Observation Probability Function	Ω_i	Client Observation Probability Function
P^{sys}	System Transition Probability Matrix	P_i	Transition Probability Matrix for Client i
π_t	System Belief at time t	$b_{i,t}$	Client i Belief at time t

In DQNs, a neural network serves as the function approximator to predict Q-values. This addition allows for better performance in high-dimensional state spaces, wherein maintaining a regular Q-table would be difficult or infeasible. In addition to applying a neural network, the DQN also incorporates experience replay; for normal MDPs, the agent's experiences, (s, a, r, s') are stored in memory which are then sampled to update the network to further stabilize training [110].

A drawback of traditional DQNs, however, is their inability to account for partially observable states. In order to improve upon their performance for application in POMDPs, DRQNs were introduced; these networks add a recurrent, long-short term memory (LSTM) to the architecture. As opposed to traditional, fully observable environments, the DRQN now inputs the observations from the environment and maintains a belief of being in a certain state. The addition of the recurrent layer allows the network to maintain information across previous time steps. This addition can better discern the underlying, hidden state. The output of the DRQN are the actions as described in section 3.2.3. The weighting scheme for the edge clients is implemented corresponding to the action provided by the DRQN, and the process repeats as described in Figure 3.4.

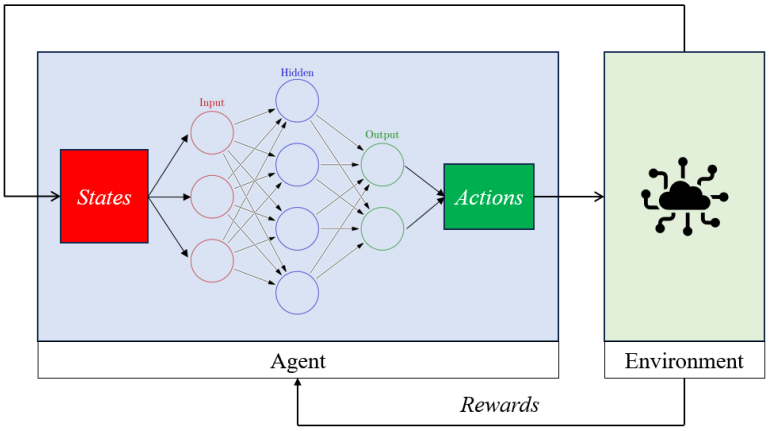


Figure 3.4: DRQN within the context of a POMDP. The DRQN inputs observations from the environment as the true underlying state is not fully observable and outputs actions for interaction with the environment.

3.2.5 Experimental Design

In order to analyze the effectiveness of our framework's approach, we have analyzed three primary experiments to determine the effectiveness of our framework against the baseline, benchmark federated learning model. We analyze our federated learning model's performance on two, three, and four edge clients across 25, 50, and 75 rounds of training. In addition, we also provide a sensitivity analysis in our two-client scenarios with different transition probability matrices and reward structures. Our experiments are described in three primary tables separated by the number of clients in each federated learning model, as described in Figure 3.5.

3.3 Experimental Results

3.3.1 Performance Metrics

To determine the effectiveness of our FLOWS framework, we apply five common metrics: accuracy, precision, recall, F1-score, and geometric mean (G-Mean) which are combinations of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The accuracy metric is calculated by $\frac{TP+TN}{TP+TN+FP+FN}$. Precision instead focuses on reducing the number of false positives and is calculated as $\frac{TP}{TP+FP}$. Recall prioritizes reduction of false negatives instead and is calculated as $\frac{TP}{TP+FN}$. F1-score and G-Mean are calculated as $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$ and $\sqrt{\frac{TP}{TP+FN}\frac{TN}{FP+TN}}$, respectively. We prioritize the G-Mean metric to determine the best model as this metric considers the performance of the model across all classes, not just the majority class. In addition, we compare our results of our best two models using a paired t-test using 10-fold cross validation. Our federated learning model utilizes Tensorflow [2]. We run our experiments on an Intel Xeon processor with 128GB of RAM.

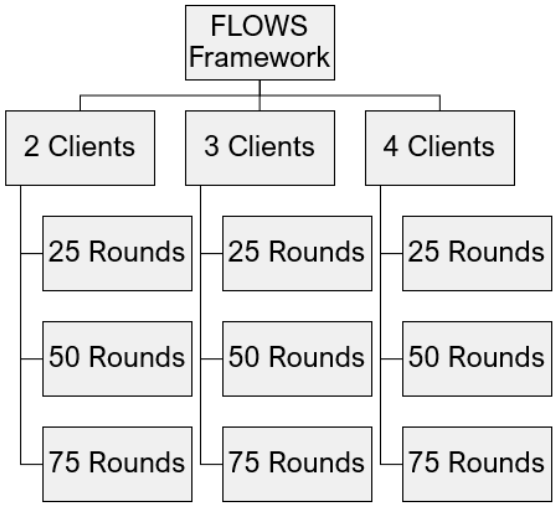


Figure 3.5: Experimental design for our FLOWS framework.

3.3.2 Performance

This section provides insight into both our baseline, or *benchmark* federated learning, and novel framework models' performance across our two-, three-, and four-client scenarios when applied in 25, 50, and 75 total communication rounds.

2-Client Scenario

Table 3.3 presents our results when applying our FLOWS framework approach compared with the benchmark federated learning model in a 2-client scenario across 25, 50, and 75 total communication rounds. Notably, we observe that our FLOWS framework statistically outperforms the benchmark model in greater rounds across several metrics; for rounds 50 and 75, our framework exhibited statistically higher results in terms of accuracy, recall, and G-Mean. We observed that our framework statistically outperformed our benchmark in the F1-score metric as well. The benchmark federated learning model tended to have better performance on the precision metric compared with our FLOWS framework.

Table 3.3: Results from the benchmark federated learning and our FLOWS framework models using two clients on the NSL-KDD intrusion detection dataset. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated Benchmark*	0.9225 (0.0133)	**0.9287 (0.0473)**	0.8942 (0.0341)	0.9094 (0.0116)	0.9180 (0.0110)
	2	*FLOWS Framework*	**0.9240 (0.0009)**	0.9112 (0.0193)	**0.9146 (0.0191)**	**0.9125 (0.0008)**	**0.9227 (0.0021)**
50	2	*Federated Benchmark*	0.9015 (0.0308)	**0.8508 (0.0652)**	0.9499 (0.0292)	0.8951 (0.0263)	0.9044 (0.0283)
	2	*FLOWS Framework*	**0.9220* (0.0096)**	0.8505 (0.0163)	**0.9973* (0.0009)**	**0.9179 (0.0092)**	**0.9280* (0.0091)**
75	2	*Federated Benchmark*	0.8981 (0.0054)	**0.9216* (0.0259)**	0.8549 (0.0336)	0.8859 (0.0047)	0.8987 (0.0037)
	2	*FLOWS Framework*	**0.9243* (0.0054)**	0.8754 (0.0260)	**0.9434* (0.0034)**	**0.9072* (0.0047)**	**0.9164* (0.0038)**

3-Client Scenario

Table 3.4 presents our results when applying our FLOWS framework compared with the benchmark federated learning model in a 3-client scenario across 25, 50, and 75 total communication rounds. We observe greater variability in results compared to the results seen in Table 3.3. Our FLOWS framework outperformed the federated learning benchmark in accuracy and precision across 25, 50, and 75 total communication rounds. The performance of our FLOWS framework, especially with respect to precision indicates that our FLOWS framework had less false positives compared to the benchmark. However, we observe that the federated learning benchmark outperformed our model in the recall metrics across the communication rounds, and had the better F1-score and G-Mean when tested on 75 communication rounds. Our FLOWS framework still outperformed our benchmark on the F1-score and G-Mean on both F1-score and G-Mean when tested on 25 and 50 communication rounds.

4-Client Scenario

Table 3.5 presents our results when applying our FLOWS framework compared with the benchmark federated learning model in a 4-client scenario across 25, 50, and 75 total communication rounds. Our FLOWS framework statistically outperformed the benchmark federated learning model on the recall metric and generally outperformed

Table 3.4: Results from the benchmark federated learning and our FLOWS framework models using three clients on the NSL-KDD intrusion detection dataset. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	3	*Federated*	0.9123	0.8889	**0.9275**	0.8936	0.9065
		Benchmark	(0.0577)	(0.0691)	**(0.1493)**	(0.1126)	(0.0932)
	3	*FLOWS*	**0.9486***	**0.9890***	0.8937	**0.9379**	**0.9407**
		Framework	**(0.0152)**	**(0.0278)**	(0.0402)	**(0.0206)**	**(0.0187)**
50	3	*Federated*	0.8987	0.8563	**0.8562**	0.8371	0.8480
		Benchmark	(0.1114)	(0.2245)	**(0.2846)**	(0.2735)	(0.2655)
	3	*FLOWS*	**0.9270**	**0.9948**	0.8341	**0.9065**	**0.9119**
		Framework	**(0.0379)**	**(0.0233)**	(0.0895)	**(0.0571)**	**(0.0504)**
75	3	*Federated*	0.9275	0.8925	**0.9519***	**0.9185**	**0.9291**
		Benchmark	(0.0287)	(0.0667)	**(0.0428)**	**(0.0269)**	**(0.0266)**
	3	*FLOWS*	**0.9282**	**0.9943***	0.8325	0.9018	0.9063
		Framework	**(0.0408)**	**(0.0226)**	(0.0969)	(0.1041)	(0.0980)

Table 3.5: Results from the benchmark federated learning and our FLOWS framework models using four clients on the NSL-KDD intrusion detection dataset. Statistically significant results are indicated by an asterisk (*).

Round	clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	4	*Federated*	0.9057	**0.8778**	0.9366	0.9011	0.8922
		Benchmark	(0.1006)	**(0.1005)**	(0.0346)	(0.0644)	(0.1708)
	4	*FLOWS*	**0.9240**	0.8525	**0.9964***	**0.9188**	**0.9305**
		Framework	**(0.0059)**	(0.0095)	**(0.0002)**	**(0.0056)**	**(0.0056)**
50	4	*Federated*	**0.9305**	**0.9150***	0.9393	**0.9240**	0.9228
		Benchmark	**(0.0750)**	**(0.0814)**	(0.0247)	**(0.0512)**	(0.1235)
	4	*FLOWS*	0.9246	0.8535	**0.9964***	0.9194	**0.9312**
		Framework	(0.0042)	(0.0067)	**(0.0002)**	(0.0040)	**(0.0040)**
75	4	*Federated*	**0.9422**	**0.9339***	0.9414	**0.9355**	**0.9361**
		Benchmark	**(0.0633)**	**(0.0716)**	(0.0204)	**(0.0448)**	**(0.1023)**
	4	*FLOWS*	0.9250	0.8540	**0.9964***	0.9197	0.9315
		Framework	(0.0029)	(0.0047)	**(0.0002)**	(0.0028)	(0.0028)

the benchmark federated learning model on the G-Mean metric. This result thus indicates that our FLOWS framework has a greater ability to reduce the total number of false negatives; that is, our FLOWS framework reduces the number of successful attacks being labeled as benign traffic. We observe that our FLOWS framework performs better than the benchmark model with four clients during a smaller number of rounds, especially with respect to the F1-Score and Accuracy metrics. Our benchmark federated learning model was superior in the precision metric; however, we consider the greater performance on the recall metric as this metric indicates that FLOWS was able to accurately discern malicious traffic.

Sensitivity Analysis

In this section, we perform a sensitivity analysis for our two client scenarios. Specifically, we conduct three sensitivity analyses: one comparing varying the TPMs of the federated learning clients so that they are resistant to changing states akin to those seen in zero-day clients, one comparing wherein one client is more likely to be attacked and stay attacked while the other is more likely to recover and stay in the normal state, and one with partial rewards for partially correct classification of edge clients for our RL agent. In order to simulate these clients, we modify the TPM of each client in our federated learning system. Through the modification of each of the client's TPMs, we can provide a means to emulate actual clients present on a computer network. We apply for our sensitivity analysis a system of two edge clients. An overview of the TPMs of our edge clients along with a realistic representation of each node is indicated in Table 3.6. We provide more numerical results with different client and reward combinations in Appendix B.

Table 3.6: Mathematical TPMs and the corresponding clients in realistic networking settings.

Mathematical TPM	Determined by dataset sample (TPM $\approx \begin{bmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{bmatrix}$)	TPM = $\begin{bmatrix} 0.9 & 0.1 \\ 0.9 & 0.1 \end{bmatrix}$	TPM = $\begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}$	TPM = $\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$
Client Type	Ordinary Client	Resilient Client	Vulnerable Client	Zero-Day Client
Long Description	This TPM corresponds to clients that have average security and existing recovery mechanisms. While it is not immune to attacks, it can transition back to normal once attacked.	This TPM corresponds to clientsthat are more resilient to attacks. These clients tend to stay in the normal state; once attacked, they have a high probability of being remediated.	This TPM corresponds to clients that are more vulnerable to attacks. These clients are less likely to stay in the normal state and are more likely to be attacked and stay attacked.	This TPM corresponds to clients that have a zero-day vulnerability. The client mathematically tends to stay in a normal state unless they are affected by a zero-day attack. Since these attacks are likelier to go unnoticed, the probability of the client transitioning back to normal and being remediated is low.

Clients Resistant to Change (Zero-day Attacked Clients) Here, we consider the scenario wherein the clients are unlikely to change states one they are in a certain state. Specifically, we modify the client TPMs so that $P_{0,1} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$; both edge clients are resistant to change once either being in the normal or attacked states. This sensitivity analysis corresponds a use-case in which edge clients might have strong defenses as they are reluctant to change states, but once compromised, they are unable to be remediated successfully. This corresponds to zero-day compromised devices; most operate nominally, but once attacked with a zero-day, these devices stay in the attacked state because of the inability to detect most zero-day breaches. We provide graphical results of this sensitivity analysis in Figure 3.6 and also provide tabular results of our sensitivity analysis in Table 3.7.

We observe graphically in this scenario that the belief of our clients being anomalous increases rapidly as both clients enter a state of being attacked. Our RL algorithm struggles, however, to accurately determine the true state of the underlying system when the belief of the clients being in the attacked state is still high as shown in rounds 12 to 15 in our graphs. The cumulative accuracy and G-Mean for the sensitivity analysis for our FLOWS framework outperforms the benchmark federated learning system on the depicted 25 rounds.

Table 3.7 provides a tabular overview of results comparing benchmark federated learning to our FLOWS sensitivity analysis with clients resistant to change. As observed through the table, we find that our sensitivity analysis tends to perform better on all metrics compared to the benchmark federated learning when tested on 25 rounds. The performance of our FLOWS framework generally decreases as the number of rounds increases; the benchmark model tends to statistically outperform our FLOWS framework with greater rounds. We do observe, however, that the recall metric tends to increase over time compared to the other performance metrics for our FLOWS framework model. This observation indicates that in this context, the FLOWS framework correctly identifies more true positives but also tends to classify

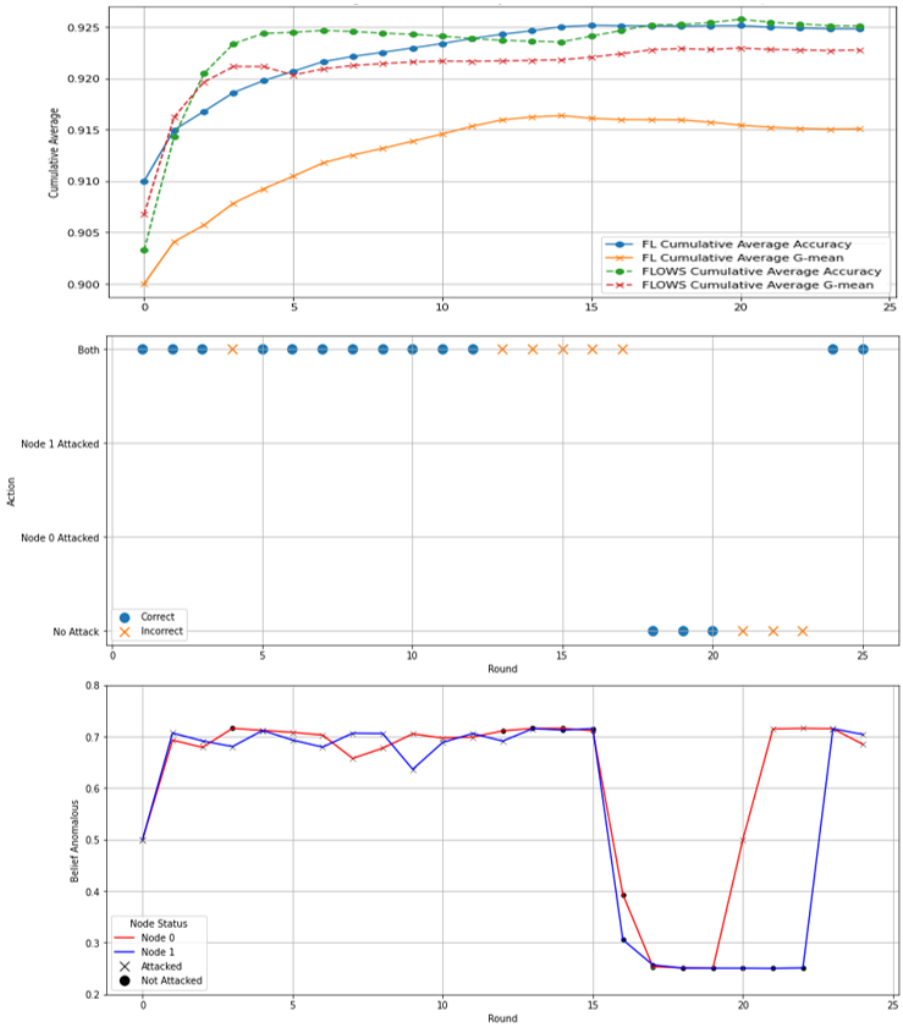


Figure 3.6: Belief vs underlying state, DRQN action, and cumulative accuracy and G-Mean comparison graph for our sensitivity analysis wherein clients are resistant to changing states.

Table 3.7: Results from the benchmark federated learning and our FLOWS framework using a sensitivity analysis wherein underlying client states are resistant to change for two clients on the NSL-KDD intrusion detection dataset. We compare our sensitivity analysis to the benchmark federated learning model. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated Benchmark*	0.9225 (0.0133)	0.9287 (0.0473)	0.8942 (0.0341)	0.9094 (0.0116)	0.9180 (0.0110)
	2	*FLOWS-Sensitivity Resistant Clients*	**0.9267 (0.0063)**	**0.9338 (0.0296)**	**0.8953 (0.0253)**	**0.9134 (0.0066)**	**0.9222 (0.0060)**
50	2	*Federated Benchmark*	**0.9015 (0.0308)**	0.8508 (0.0652)	**0.9499 (0.0292)**	**0.8951 (0.0263)**	**0.9044 (0.0283)**
	2	*FLOWS-Sensitivity Resistant Clients*	0.8834 (0.0723)	**0.8699 (0.1344)**	0.9060 (0.0578)	0.8770 (0.0567)	0.8785 (0.0694)
75	2	*Federated Benchmark*	**0.8981* (0.0054)**	**0.9216* (0.0259)**	0.8549 (0.0336)	**0.8859* (0.0047)**	**0.8987* (0.0037)**
	2	*FLOWS-Sensitivity Resistant Clients*	0.8293 (0.0970)	0.7827 (0.1656)	**0.9353* (0.0629)**	0.8362 (0.0742)	0.8239 (0.0962)

more false positives over time. Thus, changing the clients' TPMs to have them more resistant to change leads to more false positive classifications.

Vulnerable and Resilient Client Here, we cover the two-client scenario of one client having a greater propensity to be attacked and stay attacked, that is, the TPM for one client is $P_0 = \begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}$. The other client instead has a greater probability of staying in the normal state and transitioning back to normal once attacked; the TPM for this client is represented as $P_1 = \begin{bmatrix} 0.9 & 0.1 \\ 0.9 & 0.1 \end{bmatrix}$. This scenario realistically corresponds to a scenario wherein one client is more vulnerable to attack, while another client has controls and security mechanisms built in place to make it more resilient to attacks. Our graphical results are described in Figure 3.7, and we also provide tabular results of our sensitivity analysis in Table 3.8.

We observe that our non-vulnerable client, node 0, tends to remain in the non-attacked, normal state while our vulnerable client, node 1, tends to remain in the attacked state. Respectively, our RL agent acts by only characterizing the system as having one client, node 1, under attack, even if both clients are normal or under attack. We still observe better performance on our FLOWS framework as opposed to the benchmark federated learning model.

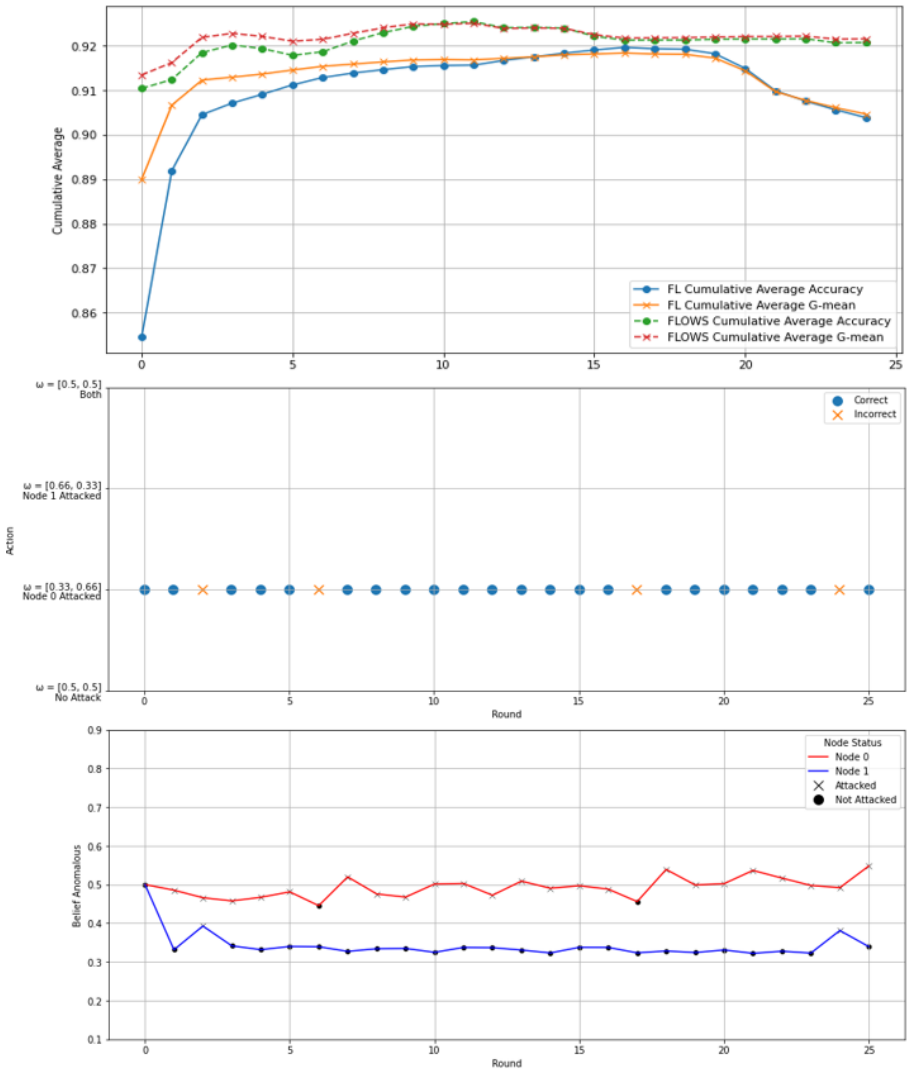


Figure 3.7: Belief vs underlying state, DRQN action, and cumulative accuracy and G-Mean comparison graph for our sensitivity analysis wherein one client is more likely to be vulnerable to attacks and remain attacked and the other client is more likely to be resilient to attacks and remain in the normal state.

Table 3.8: Results from the benchmark federated learning and our FLOWS framework using a sensitivity analysis wherein one client has a higher probability of being attacked and staying attacked while the other client has a higher probability of transitioning to normal and staying in the normal state for two clients on the NSL-KDD intrusion detection dataset. We compare our sensitivity analysis to the benchmark federated learning model. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated*	0.9225	0.9287	**0.8942***	**0.9094**	**0.9180**
		Benchmark	(0.0133)	(0.0473)	**(0.0341)**	**(0.0116)**	**(0.0110)**
	2	*FLOWS-Sensitivity*	**0.9226**	**0.9688***	0.8493	0.9037	0.9108
		Vulnerable & Resilient	**(0.0156)**	**(0.0223)**	(0.0508)	(0.0244)	(0.0223)
50	2	*Federated*	0.9015	0.8508	**0.9499***	0.8951	0.9044
		Benchmark	(0.0308)	(0.0652)	**(0.0292)**	(0.0263)	(0.0283)
	2	*FLOWS-Sensitivity*	**0.9243***	**0.9690***	0.8530	**0.9064**	**0.9132**
		Vulnerable & Resilient	**(0.0115)**	**(0.0211)**	(0.0392)	**(0.0178)**	**(0.0168)**
75	2	*Federated*	0.8981	0.9216	**0.8549**	0.8859	0.8987
		Benchmark	(0.0054)	(0.0259)	**(0.0336)**	(0.0047)	(0.0037)
	2	*FLOWS-Sensitivity*	**0.9209***	**0.9613***	0.8533	**0.9021**	**0.9099**
		Vulnerable & Resilient	**(0.0180)**	**(0.0316)**	(0.0564)	**(0.0281)**	**(0.0257)**

Our results compare benchmark federated learning to our FLOWS sensitivity analysis with one vulnerable client; the client is more prone to attacks and stays in the attacked state and another client that is resilient, that is, the client is more prone to being in the normal state and stays in the normal state. For this sensitivity analysis, we find that the performance of our FLOWS framework generally tends to increase over rounds compared to the previous sensitivity analysis. Our FLOWS framework also tends to statistically outperform the federated learning benchmark on both the accuracy and precision metrics and manages to perform better compared with the benchmark federated learning model on the F1 and G-Mean scores over greater rounds. We do find that federated learning benchmark has greater recall compared with our FLOWS framework; however, our model still improves on its recall scores with more rounds. Our results indicate that our FLOWS framework thus manages to accurately classify cyber threats compared with the benchmark model.

Partial Rewards Here, we consider the scenario wherein we award our RL agent partial rewards for partially correct actions. Recall previously we have a reward

structure of:

$$R_t(s,a) = \begin{cases} R_{TN}, R_{TP} = 1, a = s \\ R_{FP}, R_{FN} = -1, a \neq s \end{cases}$$

in which our RL agent was rewarded only if it managed to fully determine the underlying system state. Our original reward structure did not consider instances in which it classified one of the two edge clients correctly. As a result, we now consider this reward structure for our two-client scenario:

$$R_t(s,a) = \begin{cases} R_t = 3, a = s \\ R_t = 1, a \in \{0,3\} \text{ and } s \in \{1,2\} \\ R_t = 1, a \in \{1,2\} \text{ and } s \in \{0,3\} \\ R_t = -1 \text{ otherwise} \end{cases}$$

Recall that actions $a \in \{1,2\}$ correspond to marking the system as having one client normal, one client attacked. If the true underlying state has either no clients attacked, $s = 0$, or all clients attacked, $s = 3$, then our RL agent has classified at least one client correctly. Respectively, if the true underlying state is either both clients normal, $s = 0$, or both clients attacked, $s = 3$, then if our RL agent takes action $a = 1$ or $a = 2$, it has classified at least one of the edge clients correctly. This scenario is meant to provide more leeway for our RL agent; there is less punishment if the agent does not accurately discern the true underlying state of the federated learning system. We provide graphical results of this sensitivity analysis in Figures 3.8 and 3.9. Figure 3.8 provides a comparison of cumulative rewards, while Figure 3.9 provides a plot of actions taken by the RL agent and performance compared to the benchmark federated learning model. We also provide tabular results in Table 3.9.

From our sensitivity analysis, we find that when modifying the reward structure, we observe that our RL agent obtains a lower amount of cumulative rewards over the number of episodes compared to the unmodified reward structure. Interestingly, in

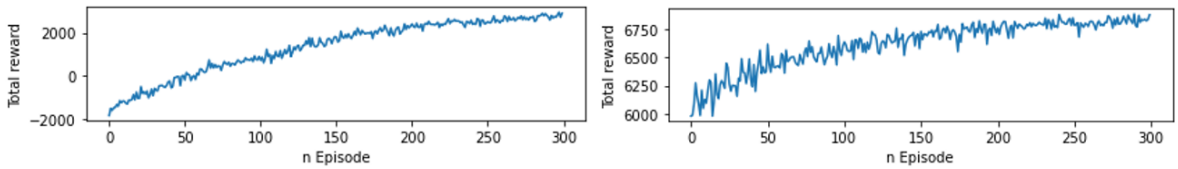


Figure 3.8: Comparison of RL agent reward performance with a normal rewards structure (left) versus modified reward structure (right). While the RL agent starts at a higher reward level with the modified reward structure, the cumulative growth in rewards is lower.

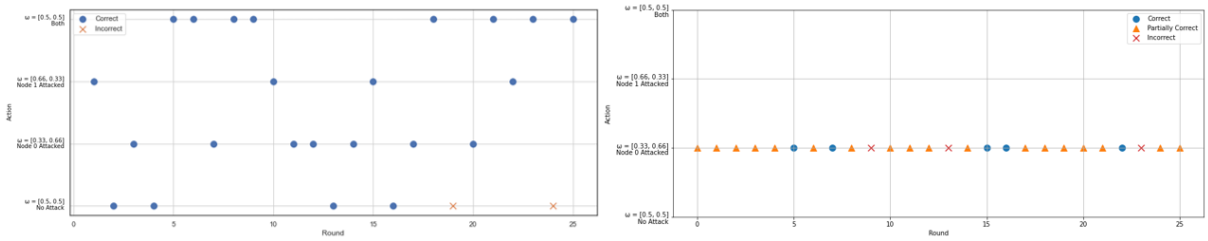


Figure 3.9: Comparison of RL agent action with a normal rewards structure (left) versus modified reward structure (right). While the RL agent with normal rewards tends to select a variety of actions, the RL agent with modified rewards selects only one.

Table 3.9: Results from the benchmark federated learning and our FLOWS framework using a sensitivity analysis with partial rewards for partially correct classification for two clients on the NSL-KDD intrusion detection dataset. We compare our sensitivity analysis to the benchmark federated learning model. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated*	**0.9225**	**0.9287**	0.8942	**0.9094**	**0.9180**
		Benchmark	**(0.0133)**	**(0.0473)**	(0.0341)	**(0.0116)**	**(0.0110)**
	2	*FLOWS-Sensitivity*	0.9105	0.8859	**0.9195**	0.8995	0.9102
		Different Rewards	(0.0221)	(0.0657)	**(0.0377)**	(0.0185)	(0.0173)
50	2	*Federated*	**0.9015**	**0.8508**	**0.9499**	**0.8951**	**0.9044**
		Benchmark	**(0.0308)**	**(0.0652)**	**(0.0292)**	**(0.0263)**	**(0.0283)**
	2	*FLOWS-Sensitivity*	0.8888	0.8405	0.9360	0.8813	0.8910
		Different Rewards	(0.0384)	(0.0857)	(0.0399)	(0.0318)	(0.0340)
75	2	*Federated*	**0.8981**	**0.9216***	0.8549	0.8859	**0.8987**
		Benchmark	**(0.0054)**	**(0.0259)**	(0.0336)	(0.0047)	**(0.0037)**
	2	*FLOWS-Sensitivity*	0.8952	0.8559	**0.9286***	**0.8865**	0.8964
		Different Rewards	(0.0368)	(0.0837)	**(0.0389)**	**(0.0305)**	(0.0322)

a modified reward structure, our RL agent grows its reward only by an amount of roughly 750; the agent starts at a reward level of 6000 and grows to 6750 at the end of 300 episodes with 200 iterations per episode. Whereas in the unmodified reward structure, while the RL agent starts at roughly a level of -2000 for rewards, the agent's cumulative rewards grow to an amount of nearly 2000. This observation demonstrates a greater growth in rewards when using the unmodified reward structure as opposed to the modified reward structure.

Additionally, when applying this modified reward structure, we find that our RL agent tends to take only a single action instead of changing actions. This behavior is indicative of the modified reward structure; as the agent can still obtain a partial reward for not being correct, there could be less incentive for the agent to change its chosen actions.

As observed through the table, we find that our FLOWS framework is outperformed by the benchmark federated learning model in most performance metrics. We find that initially, with 25 rounds, our model performs better with respect to recall; however, when tested on 50 rounds the benchmark federated learning has the best performance. We do observe that for 75 rounds, our novel framework outperforms the federated learning benchmark on the F1 metric and statistically outperforms the benchmark on the recall metric.

3.4 Discussion

Our FLOWS framework tended to outperform the benchmark federated learning model primarily in the metrics of accuracy, F1-score, and G-Mean. Notably, we observe that our FLOWS framework additionally has a lower standard deviation of scores as opposed to the the values observed in the benchmark federated learning model. In addition, we tend to observe that the increase of number of clients correspond to a decrease of the performance of our benchmark federated learning model; there is an inverse relationship. This trend was not as prevalent when

utilizing our novel framework for analysis, indicating that our novel framework tends to perform more persistently compared to the benchmark models. We do observe that our FLOWS framework outperforms our federated learning benchmark on the recall metric; this observation indicates that our novel framework is statistically better suited to detecting threats as opposed to the benchmark federated learning model.

Furthermore, we observe accurate results from our reinforcement learning agent using the DRQN algorithm. Our results indicate that our RL agent tends to correctly assess the underlying state of the client, whether in an attacked state or normal. Additionally, we do observe that our DRQN tends to accurately classify edge clients given the client beliefs as demonstrated in graphical results in our sensitivity analyses. Because of the accurate discernment of both underlying client and full federated learning system states, we observe better performance on the second stage of our FLOWS framework leading to better classification of network attacks.

Our sensitivity analyses utilizing a two-client scenario also provided insights into our model's performance. Performance of our FLOWS framework demonstrate that our RL agent tends to struggle in situations wherein belief of the client does not reflect the true underlying state of the client. When the belief does not accurately reflect the true underlying client state, our RL agent errs in predicting the correct state. These errors were reflected in the numerical performance of the sensitivity analysis; while initially outperforming the federated learning benchmark on 25 rounds, over more rounds, the benchmark outperforms the FLOWS framework. These errors were more prevalent in the scenario in which our RL agent was partially awarded for partially correct classification of the underlying federated learning state; the RL agent tended to take only one action which was partially correct as it could obtain some rewards. Both of these sensitivity analyses demonstrated an increase in the recall metric for our FLOWS framework model. These results indicate that for these sensitivity analyses, increases in the number of rounds leads to reduction of false negatives but an increase of false positives. While the increase of false-positives could cause added strain on

defenders, ultimately, the decrease in false negatives would lead to a decrease in successful exploits or attacks.

For our second sensitivity analysis wherein one of our clients was vulnerable and stays attacked and another client is resilient and stays in the normal state, we observe significantly better performance for our FLOWS framework on metrics such as accuracy and precision. Note that in this sensitivity analysis, the TPMs for each client had a greater probability of being in state s_0 or s_1 corresponding to normal or attacked for the vulnerable and resilient clients, respectively. Thus, our RL agent managed to determine an action that allowed for the best weighting for our edge clients as they remained in a certain state.

Overall, we generally observe that our FLOWS framework has the ability to accurately discern the true underlying state of the federated learning system, and exhibited better performance even with changes in the number of edge clients and the number of rounds. Our FLOWS framework thus was more likely to demonstrate better performance across different scenarios and in certain sensitivity analyses. Formulating the federated network intrusion detection problem instead as a two-stage POMDP yields improved performance compared to non-POMDP federated learning intrusion detection.

3.5 Conclusion

Reliable cybersecurity requires the introduction of technologies that are meant to maintain the confidentiality, integrity, and availability of organizational systems. Recently, several efforts to maintain privacy in addition to confidentiality, integrity, and availability of many internetworked devices have extended into machine learning. Federated learning, a recent development in the field of machine learning which involves training of a centralized machine learning model on edge clients, has shown promise in the field of cyber network intrusion detection. The federated learning averaging algorithm does have the drawback of incorporating all the edge clients into

the global, centralized model without considering client health or status thus leading to potential concerns with the performance of the centralized model in potentially adversarial environments.

In our work, we provided a novel method to weight edge clients based upon each client's performance by formulating the problem as a POMDP. Our FLOWS framework outperforms benchmark federated learning when tested across scenarios involving two, three, and four edge clients in 25, 50, and 75 communication rounds. We observe improvements in metrics close to to 4% in the G-Mean metric when applying our FLOWS framework. We additionally performed three sensitivity analyses which capture more realistic networking scenarios for our FLOWS framework: 1) an analysis wherein our clients were resistant to change, 2) an analysis wherein one client was vulnerable and one was resilient, and 3) an analysis wherein our RL agent had a different reward structure. Our FLOWS framework still manages to outperform the benchmark federated learning model in some of our sensitivity analyses.

There are several potential research directions that can be analyzed for future works. While our framework tends to work in cases where the client number is limited, we acknowledge that continual increases in the number of clients will lead to an exponential growth of total actions that our RL agent must take. For future works, we can apply the usage of RL algorithms that are more situated for continuous action spaces, such as the asynchronous advantage actor critic algorithm [77]. In addition, we can analyze the usage of targeted attacks against our federated learning models through applying sophisticated adversarial machine learning algorithms. We can also apply our framework to other problems in cybersecurity, such as for malware detection. Ultimately, our work provides a promising and generalizable approach for cybersecurity protection across a variety of computer networks with several clients.

Bibliography

[1] M. Aamir and S. M. A. Zaidi. Ddos attack detection with feature engineering and machine learning: the framework and performance evaluation. *International Journal of Information Security*, 18(6):761–785, 2019. doi: https://doi.org/10.1007/s10207-019-00434-1. 59

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org. 21, 109

[3] T. S. Ajani, A. L. Imoize, and A. A. Atayero. An overview of machine learning within embedded and mobile devices–optimizations and applications. *Sensors*, 21(13), 2021. ISSN 1424-8220. doi: 10.3390/s21134412. 49

[4] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In C. V. Jawahar, H. Li, G. Mori, and K. Schindler, editors, *Computer Vision – ACCV 2018*,

pages 622–637, Cham, 2019. Springer International Publishing. doi: https://doi.org/10.1007/978-3-030-20893-6_39. xix, 58, 70, 71, 81, 82, 89

[5] D. Akgun, S. Hizal, and U. Cavusoglu. A new ddos attacks intrusion detection model based on deep learning for cybersecurity. *Computers & Security*, 118: 102748, 2022. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2022.102748. 60

[6] A. Alamleh, O. S. Albahri, A. A. Zaidan, A. S. Albahri, A. H. Alamoodi, B. B. Zaidan, S. Qahtan, H. A. Alsatar, M. S. Al-Samarraay, and A. N. Jasim. Federated learning for iomt applications: A standardization and benchmarking framework of intrusion detection systems. *IEEE Journal of Biomedical and Health Informatics*, 27(2):878–887, 2023. doi: 10.1109/JBHI.2022.3167256. 96, 97

[7] T. Aldwairi, D. Perera, and M. A. Novotny. An evaluation of the performance of restricted boltzmann machines as a model for anomaly network intrusion detection. *Computer Networks*, 144:111–119, 2018. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2018.07.025. 5, 61, 63

[8] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad. Solar winds hack: In-depth analysis and countermeasures. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–7, 2021. doi: https://doi.org/10.1109/ICCCNT51525.2021.9579611. 56

[9] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar. Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020. doi: 10.1109/ACCESS.2020.3022862. 8, 15, 16, 17, 19

[10] H. Anderson. Introduction to nessus. *Retrieved from Symantec*, 2003. 16

[11] R. G. Bace. *Intrusion detection*. Sams Publishing, 2000. 97

[12] D. Baumer, J. B. Earp, and F. C. Payton. Privacy of medical records: IT implications of HIPAA. *ACM SIGCAS Computers and Society*, 30(4):40–47, 2000. 95

[13] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957. 107

[14] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. d. Hartog. Ton_iot: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets. *IEEE Internet of Things Journal*, 9(1):485–496, 2022. doi: 10.1109/JIOT.2021.3085194. 15

[15] S. Branco, A. G. Ferreira, and J. Cabral. Machine learning in resource-scarce embedded systems, fpgas, and end-devices: A survey. *Electronics*, 8(11), 2019. ISSN 2079-9292. doi: 10.3390/electronics8111289. 49

[16] G. Caminero, M. Lopez-Martin, and B. Carro. Adversarial environment reinforcement learning algorithm for intrusion detection. *Computer Networks*, 159:96–109, 2019. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2019.05.013. 2, 4, 7, 12, 52

[17] M. Catillo, A. Pecchia, and U. Villano. Cps-guard: Intrusion detection for cyber-physical systems and iot devices using outlier-aware deep autoencoders. *Computers & Security*, 129:103210, 2023. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2023.103210. 4

[18] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002. 68

[19] Z. Che, Y. Cheng, S. Zhai, Z. Sun, and Y. Liu. Boosting deep learning risk prediction with generative adversarial networks for electronic health records. In

2017 IEEE International Conference on Data Mining (ICDM), pages 787–792. IEEE, 2017. doi: https://doi.org/10.1109/ICDM.2017.93. 89

[20] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*, pages 1–5, 2018. doi: https://doi.org/10.1109/WTS.2018.8363930. 61

[21] Z. Chen, N. Lv, P. Liu, Y. Fang, K. Chen, and W. Pan. Intrusion detection for wireless edge networks based on federated learning. *IEEE Access*, 8:217463–217472, 2020. doi: 10.1109/ACCESS.2020.3041793. 96, 97

[22] B.-Y. Choi and S. Bhattacharyya. Observations on cisco sampled netflow. *SIGMETRICS Perform. Eval. Rev.*, 33(3):18–23, dec 2005. ISSN 0163-5999. doi: https://doi.org/10.1145/1111572.1111579. 57, 64

[23] D. Chou and M. Jiang. A survey on data-driven network intrusion detection. *ACM Computing Surveys (CSUR)*, 54(9):1–36, 2021. 94, 97

[24] I. Covert, S. M. Lundberg, and S.-I. Lee. Understanding global feature contributions with additive importance measures. *Advances in Neural Information Processing Systems*, 33:17212–17223, 2020. 4, 15

[25] M. C. Dacier, H. König, R. Cwalinski, F. Kargl, and S. Dietrich. Security challenges and opportunities of software-defined networking. *IEEE Security & Privacy*, 15(2):96–100, 2017. doi: https://doi.org/10.1109/MSP.2017.46. 55, 67

[26] S. Das, A. M. Mahfouz, D. Venugopal, and S. Shiva. Ddos intrusion detection through machine learning ensemble. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 471–477, 2019. doi: https://doi.org/10.1109/QRS-C.2019.00090. 61, 63

[27] A. Dimara, V.-G. Vasilopoulos, A. Papaioannou, S. Angelis, K. Kotis, C.-N. Anagnostopoulos, S. Krinidis, D. Ioannidis, and D. Tzovaras. Self-healing of

semantically interoperable smart and prescriptive edge devices in IoT. *Applied Sciences*, 12(22), 2022. ISSN 2076-3417. doi: 10.3390/app122211650. URL https://www.mdpi.com/2076-3417/12/22/11650. 96

[28] Y. Ding and Y. Zhai. Intrusion detection system for nsl-kdd dataset using convolutional neural networks. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, page 81–85, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450366069. doi: https://doi.org/10.1145/3297156.3297230. 61, 63

[29] V. Dixit, R. Selvarajan, T. Aldwairi, Y. Koshka, M. A. Novotny, T. S. Humble, M. A. Alam, and S. Kais. Training a quantum annealing based restricted boltzmann machine on cybersecurity data. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 96:1–12, 2021. doi: https://doi.org/10.1109/TETCI.2021.3074916. 61, 63

[30] S. Dowling, M. Schukat, and E. Barrett. Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware. *Journal of Cyber Security Technology*, 2(2):75–91, 2018. doi: 10.1080/23742917.2018.1495375. 7

[31] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. Inter-dataset generalization strength of supervised machine learning methods for intrusion detection. *Journal of Information Security and Applications*, 54:102564, 2020. ISSN 2214-2126. doi: https://doi.org/10.1016/j.jisa.2020.102564. 60

[32] R. Elderman, L. J. Pater, A. S. Thie, M. M. Drugan, and M. A. Wiering. Adversarial reinforcement learning in a cyber security simulation. In *ICAART (2)*, pages 559–566, 2017. doi: 10.5220/0006197105590566. 6

[33] D. Farrugia, C. Zerafa, T. Cini, B. Kuasney, and K. Livori. A real-time prescriptive solution for explainable cyber-fraud detection within the igaming industry. *Sn computer science*, 2(3):215, 2021. 96, 99

[34] A. Fatani, A. Dahou, M. Abd Elaziz, M. A. A. Al-qaness, S. Lu, S. A. Alfadhli, and S. S. Alresheedi. Enhancing intrusion detection systems for iot and cloud environments using a growth optimizer algorithm and conventional neural networks. *Sensors*, 23(9), 2023. ISSN 1424-8220. doi: 10.3390/s23094430. 7

[35] T. M. Fernández-Caramés and P. Fraga-Lamas. Teaching and learning iot cybersecurity and vulnerability assessment with shodan through practical use cases. *Sensors*, 20(11):3048, 2020. 5

[36] M. A. Ferrag, L. Maglaras, A. Ahmim, M. Derdour, and H. Janicke. Rdtids: Rules and decision tree-based intrusion detection system for internet-of-things networks. *Future internet*, 12(3):44, 2020. 67

[37] E. Ficke, K. M. Schweitzer, R. M. Bateman, and S. Xu. Characterizing the effectiveness of network-based intrusion detection systems. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 76–81, 2018. doi: https://doi.org/10.1109/MILCOM.2018.8599700. 56

[38] R. Ganesan, S. Jajodia, A. Shah, and H. Cam. Dynamic scheduling of cybersecurity analysts for minimizing risk using reinforcement learning. *ACM Trans. Intell. Syst. Technol.*, 8(1), jul 2016. ISSN 2157-6904. doi: 10.1145/2882969. 6

[39] M. C. Ghanem and T. M. Chen. Reinforcement learning for efficient network penetration testing. *Information*, 11(1), 2020. ISSN 2078-2489. doi: 10.3390/info11010006. 5

[40] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 62, 69

[41] P. Goyal and A. Goyal. Comparative study of two most popular packet sniffing tools-tcpdump and wireshark. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 77–81, 2017. doi: https://doi.org/10.1109/CICN.2017.8319360. 57

[42] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. doi: https://doi.org/10.5555/3295222.3295327. 62

[43] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015. 12, 107

[44] R. Hertzog, J. O'Gorman, and M. Aharoni. Kali linux revealed. *Mastering the Penetration Testing Distribution*, 2017. 15

[45] G. E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012. 57, 72, 73, 74

[46] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 16(4): 2037–2064, 2014. 55

[47] C. J. Hoofnagle, B. Van Der Sloot, and F. Z. Borgesius. The European Union General Data Protection Regulation: what it is and what it means. *Information & Communications Technology Law*, 28(1):65–98, 2019. 95

[48] Z. Hu, R. Beuran, and Y. Tan. Automated penetration testing using deep reinforcement learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 2–10, 2020. doi: 10.1109/EuroSPW51379.2020.00010. 5

[49] Y. Hua, J. Guo, and H. Zhao. Deep belief networks and deep learning. In *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, pages 1–4, 2015. doi: 10.1109/ICAIOT.2015.7111524. 74

[50] M. J. Idrissi, H. Alami, A. El Mahdaouy, A. El Mekki, S. Oualil, Z. Yartaoui, and I. Berrada. Fed-ANIDS: Federated learning for anomaly-based network intrusion detection systems. *Expert Systems with Applications*, 234:121000, 2023. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2023.121000. URL https://www.sciencedirect.com/science/article/pii/S0957417423015026. 96, 97

[51] S. Islam. Security auditing tools: a comparative study. *International Journal of Computing Sciences Research*, 5(1):407–425, 2021. 16

[52] V. Kelli, P. Radoglou-Grammatikis, T. Lagkas, E. K. Markakis, and P. Sarigiannidis. Risk analysis of dnp3 attacks. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 351–356, 2022. doi: 10.1109/CSR54599.2022.9850291. 65, 68

[53] J. Khoury and M. Nassar. A hybrid game theory and reinforcement learning approach for cyber-physical systems security. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2020. doi: 10.1109/NOMS47738.2020.9110453. 6

[54] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22, 2019. 2

[55] J.-Y. Kim, S.-J. Bu, and S.-B. Cho. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. *Information Sciences*, 460-461:83–102, 2018. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2018.04.092. 5

[56] K. Kirkpatrick. Software-defined networking. *Commun. ACM*, 56(9):16–19, sep 2013. ISSN 0001-0782. doi: https://doi.org/10.1145/2500468.2500473. 55

[57] R. Kokila, S. Thamarai Selvi, and K. Govindarajan. Ddos detection and analysis in sdn-based environment using support vector machine classifier. In *2014 Sixth International Conference on Advanced Computing (ICoAC)*, pages 205–210, 2014. doi: https://doi.org/10.1109/ICoAC.2014.7229711. 59, 63

[58] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis. Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *IEEE Communications Surveys & Tutorials*, 18(1):184–208, 2016. doi: 10.1109/COMST.2015.2402161. 7

[59] M. V. Kotpalliwar and R. Wajgi. Classification of attacks using support vector machine (svm) on kddcup'99 ids database. In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages 987–990, 2015. doi: https://doi.org/10.1109/CSNT.2015.185. 59

[60] V. Krishnamurthy. *Partially observed Markov decision processes*. Cambridge university press, 2016. 104

[61] M. L. Labs. KDD-Cup 99. https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, October 1998. 100

[62] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 10, 11

[63] D. Limon-Cantu and V. Alarcon-Aquino. Multiresolution dendritic cell algorithm for network anomaly detection. *PeerJ Computer Science*, 7:e749, 2021. 61

[64] Z. Lin, Y. Shi, and Z. Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection, 2018. URL https://arxiv.org/abs/1809.02077. 62

[65] Z. Lin, Y. Shi, and Z. Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 79–91. Springer, 2022. doi: https://doi.org/10.1007/978-3-031-05981-0_7. 71, 89

[66] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. doi: https://doi.org/10.1109/ICDM.2008.17. 75, 81

[67] Z. Liu, A. Khojandi, X. Li, A. Mohammed, R. L. Davis, and R. Kamaleswaran. A machine learning–enabled partially observable Markov decision process framework for early sepsis prediction. *INFORMS Journal on Computing*, 34 (4):2039–2057, 2022. 96, 99

[68] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas. Application of deep reinforcement learning to intrusion detection for supervised problems. *Expert Systems with Applications*, 141:112963, 2020. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2019.112963. 7

[69] R. Maeda and M. Mimura. Automating post-exploitation with deep reinforcement learning. *Computers & Security*, 100:102108, 2021. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2020.102108. 5

[70] A. Maleki, M. Hossain, J.-P. Georges, E. Rondeau, and T. Divoux. An sdn perspective to mitigate the energy consumption of core networks–géant2. In *International SEEDS conference*, pages 233–244, 2017. xix, 56

[71] R. Masri and M. Aldwairi. Automated malicious advertisement detection using virustotal, urlvoid, and trendmicro. In *2017 8th International Conference on Information and Communication Systems (ICICS)*, pages 336–341, 2017. doi: https://doi.org/10.1109/IACS.2017.7921994. 7, 56

[72] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. 95, 97, 98, 103

[73] L. R. Medsker and L. Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001. 12

[74] J. Meira, R. Andrade, I. Praça, J. Carneiro, V. Bolón-Canedo, A. Alonso-Betanzos, and G. Marreiros. Performance evaluation of unsupervised techniques in cyber-attack anomaly detection. *Journal of Ambient Intelligence and Humanized Computing*, 11:4477–4489, 2019. doi: https://doi.org/10.1007/s12652-019-01417-9. 61, 63

[75] L. Mhamdi, D. McLernon, F. El-moussa, S. A. Raza Zaidi, M. Ghogho, and T. Tang. A deep learning approach combining autoencoder with one-class svm for ddos attack detection in sdns. In *2020 IEEE Eighth International Conference on Communications and Networking (ComNet)*, pages 1–6, 2020. doi: https://doi.org/10.1109/ComNet47917.2020.9306073. 61, 63

[76] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013. 12, 107

[77] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016. 125

[78] E. A. Morse and V. Raval. PCI DSS: Payment card industry data security standards in context. *Computer Law & Security Review*, 24(6):540–554, 2008. 95

[79] N. Moustafa and J. Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015. doi: https://doi.org/10.1109/MilCIS.2015.7348942. 7, 57

[80] N. Moustafa, M. Ahmed, and S. Ahmed. Data analytics-enabled intrusion detection: Evaluations of ton_iot linux datasets. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 727–735, 2020. doi: 10.1109/TrustCom50675.2020.00100. 15

[81] N. Moustafa, M. Keshky, E. Debiez, and H. Janicke. Federated ton_iot windows datasets for evaluating ai-based security applications. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 848–855, 2020. doi: 10.1109/TrustCom50675.2020.00114. 15

[82] National Academy of Engineering. Securing cyberspace, 2021. URL http://www.engineeringchallenges.org/9042.aspx. 55

[83] NIST. NIST CVSS severity distribution over time. https://nvd.nist.gov/general/visualizations/\vulnerability-visualizations/cvss-severity-distribution-over-time, 2023. Accessed: 2024-02-02. 94

[84] A. Orebaugh and B. Pinkard. *Nmap in the enterprise: your guide to network scanning*. Elsevier, 2011. 16

[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 21

[86] S. Perumalla, S. Chatterjee, and A. S. Kumar. Modelling of oppositional aquila optimizer with machine learning enabled secure access control in internet of drones environment. *Theoretical Computer Science*, 941:39–54, 2023. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2022.08.019. 8

[87] M. S. Pervez and D. M. Farid. Feature selection and intrusion classification in nsl-kdd cup 99 dataset employing svms. In *The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014)*, pages 1–6, 2014. doi: https://doi.org/10.1109/SKIMA.2014.7083539. 59, 63

[88] A. Piplai, P. Ranade, A. Kotal, S. Mittal, S. N. Narayanan, and A. Joshi. Using knowledge graphs and reinforcement learning for malware analysis. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2626–2633, 2020. doi: 10.1109/BigData50022.2020.9378491. 6

[89] A. Portnoy, Y. Tirosh, and D. Hendler. Towards federated learning with Byzantine-robust client weighting. *Applied Sciences*, 12(17), 2022. ISSN 2076-3417. doi: 10.3390/app12178847. 96, 98

[90] S. Pramanik, D. Samanta, M. Vinay, and A. Guha. *Cyber Security and Network Security*. John Wiley & Sons, 2022. 95, 97

[91] L. Puggini and S. McLoone. An enhanced variable selection and isolation forest based methodology for anomaly detection with oes data. *Engineering Applications of Artificial Intelligence*, 67:126–135, 2018. doi: https://doi.org/10.1016/j.engappai.2017.09.021. 75

[92] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016. 69

[93] S. Rezaei, N. Masoud, and A. Khojandi. Gaad: Gan-enabled autoencoder for real-time sensor anomaly detection and recovery in autonomous driving. *IEEE Sensors Journal*, 24(7):11734–11742, 2024. doi: 10.1109/JSEN.2024.3361460. 70, 71, 72

[94] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.06.005. 57

[95] R. Rohith, M. Moharir, G. Shobha, et al. Scapy-a powerful interactive packet manipulation program. In *2018 international conference on networking, embedded and wireless systems (ICNEWS)*, pages 1–5. IEEE, 2018. 18

[96] C. Rookard and A. Khojandi. Applying deep reinforcement learning for detection of internet-of- things cyber attacks. In *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0389–0395, 2023. doi: 10.1109/CCWC57344.2023.10099349. 7

[97] C. Rookard and A. Khojandi. A federated learning-pomdp approach for network intrusion detection. In *Preprint*, 2024. 96, 99

[98] C. Rookard and A. Khojandi. Rriot: Recurrent reinforcement learning for cyber threat detection on iot devices. *Computers & Security*, 140:103786, 2024. iii

[99] C. Rookard and A. Khojandi. Unsupervised machine learning for cybersecurity anomaly detection in traditional and software-defined networking environments. *IEEE Transactions on Network and Service Management*, 2024. iii

[100] C. Rookard, A. Crabtree, J. D. Trujillo, and A. Khojandi. Secure analytics for embedded systems through machine learning applications using trustzones. In *2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0531–0537, 2022. doi: 10.1109/UEMCON54665.2022.9965637. 72, 73, 74

[101] G. Rosa, M. Roder, D. Santos, and C. Kelton. Enhancing anomaly detection through restricted boltzmann machine features projection. *International Journal of Information Technology*, 13:49–57, 2021. doi: https://doi.org/10.1007/s41870-020-00535-4. 60, 63

[102] S. M. Ross. *Introduction to probability and statistics for engineers and scientists*. Academic press, 2020. 80

[103] K. Sadaf and J. Sultana. Intrusion detection based on autoencoder and isolation forest in fog computing. *IEEE Access*, 8:167059–167068, 2020. doi: https://doi.org/10.1109/ACCESS.2020.3022855. 61, 63

[104] M. A. Salama, H. F. Eid, R. A. Ramadan, A. Darwish, and A. E. Hassanien. Hybrid intelligent intrusion detection scheme. In *Soft computing in industrial applications*, pages 293–303. Springer, 2011. doi: https://doi.org/10.1007/978-3-642-20505-7.26. 60, 63

[105] K. Sethi, E. Sai Rupesh, R. Kumar, P. Bera, and Y. Venu Madhav. A context-aware robust intrusion detection system: a reinforcement learning-based approach. *International Journal of Information Security*, 19(6):657–678, 2020. 7

[106] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1: 108–116, 2018. doi: https://doi.org/10.5220/0006639801080116. 57, 67

[107] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, 2012. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2011.12.012. 57

[108] Z. Shu, J. Wan, D. Li, J. Lin, A. Vasilakos, and M. Imran. Security in software-defined networking: Threats and countermeasures. *Mobile Network Applications*, 21:764–776, 2016. doi: https://doi.org/10.1007/s11036-016-0676-x. 55

[109] N. B. Singh, M. M. Singh, A. Sarkar, and J. K. Mandal. A novel wide & deep transfer learning stacked gru framework for network intrusion detection. *Journal of Information Security and Applications*, 61:102899, 2021. ISSN 2214-2126. doi: https://doi.org/10.1016/j.jisa.2021.102899. 60

[110] R. S. Sutton. Reinforcement Learning: an Introduction. *A Bradford Book*, 2018. 108

[111] G. C. Swathi, G. K. Kumar, and A. S. Kumar. Estimating botnet impact on iot/ioe networks using traffic flow features. *Computers and Electrical Engineering*, 102:108209, 2022. ISSN 0045-7906. doi: https://doi.org/10.1016/j.compeleceng.2022.108209. 8

[112] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho. Deep learning approach for network intrusion detection in software defined networking. In *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 258–263, 2016. doi: https://doi.org/10.1109/WINCOM.2016.7777224. 59, 60, 63, 67

[113] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho. Deep recurrent neural network for intrusion detection in sdn-based networks. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 202–206, 2018. doi: https://doi.org/10.1109/NETSOFT.2018.8460090. 60, 63, 67

[114] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009. doi: https://doi.org/10.1109/CISDA.2009.5356528. 7, 57, 64, 65, 87, 100

[115] D. Tax and R. Duin. Support vector data description. *Machine Learning*, 54: 45–66, 2004. doi: https://doi.org/10.1023/B:MACH.0000008084.60811.49. 75

[116] U. Timalsina and K. Gurung. Metasploit framework with kali linux. *Technical Report*, 2015. 17

[117] J. Utter. Learnings from the colonial pipeline cyberattack: focus on the 98% of attacks, not the 2%!, 2021. URL https://www.securitymagazine.com/articles/95190. 55

[118] M. Wang, Y. Lu, and J. Qin. A dynamic mlp-based ddos attack detection method using feature selection and feedback. *Computers & Security*, 88:101645, 2020. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.101645. 4, 59

[119] S. Wang, D. Xu, and S. Yan. Analysis and application of wireshark in tcp/ip protocol teaching. In *2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT)*, volume 2, pages 269–272, 2010. doi: https://doi.org/10.1109/EDT.2010.5496372. 57

[120] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning, 2016. 13, 14

[121] J. Watts, F. Van Wyk, S. Rezaei, Y. Wang, N. Masoud, and A. Khojandi. A dynamic deep reinforcement learning-Bayesian framework for anomaly detection. *IEEE Transactions on Intelligent Transportation Systems*, 23(12): 22884–22894, 2022. 96, 99

[122] H. Wu and P. Wang. Node selection toward faster convergence for federated learning on non-IID data. *IEEE Transactions on Network Science and Engineering*, 9(5):3099–3111, 2022. doi: 10.1109/TNSE.2022.3146399. 96, 98

[123] S. Xia, M. Qiu, and H. Jiang. An adversarial reinforcement learning based system for cyber security. In *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 227–230, 2019. doi: 10.1109/SmartCloud.2019.00046. 6

[124] W. Xiong, E. Legrand, O. Åberg, and R. Lagerström. Cyber security threat modeling based on the mitre enterprise ATT&CK matrix. *Software and Systems Modeling*, 21(1):157–177, 2022. 102

[125] J. Xu and H. Wang. Client selection and bandwidth allocation in wireless federated learning networks: A long-term perspective. *IEEE Transactions on Wireless Communications*, 20(2):1188–1200, 2021. doi: 10.1109/TWC.2020.3031503. 96, 98

[126] D. Xuan, H. Hu, B. Wang, and B. Liu. Intrusion detection system based on rf-svm model optimized with feature selection. In *2021 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, pages 1–5, 2021. doi: https://doi.org/10.1109/CCCI52664.2021.9583206. 59

[127] Q. Yan, M. Wang, W. Huang, X. Luo, and F. R. Yu. Automatically synthesizing dos attack traces using generative adversarial networks. *International Journal of Machine Learning and Cybernetics*, 10(12):3387–3396, 2019. doi: https://doi.org/10.1007/s13042-019-00925-6. 62, 71

[128] K. Yang, J. Liu, C. Zhang, and Y. Fang. Adversarial examples against the deep learning based network intrusion detection systems. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 559–564, 2018. doi: https://doi.org/10.1109/MILCOM.2018.8599759. 62

[129] M. Yang, H. Qian, X. Wang, Y. Zhou, and H. Zhu. Client selection for federated learning with label noise. *IEEE Transactions on Vehicular Technology*, 71(2): 2193–2197, 2022. doi: 10.1109/TVT.2021.3131852. 96, 98

[130] Y. Yang, G. Xie, J. Wang, J. Zhou, Z. Xia, and R. Li. Intrusion detection for in-vehicle network by using single gan in connected vehicles. *Journal of Circuits, Systems and Computers*, 30(01):2150007, 2021. doi: https://doi.org/10.1142/S0218126621500079. 89

[131] W. Zhang, X. Wang, P. Zhou, W. Wu, and X. Zhang. Client selection for federated learning with non-IID data in mobile edge computing. *IEEE Access*, 9:24462–24474, 2021. doi: 10.1109/ACCESS.2021.3056919. 96, 98

[132] R. Zhao, Y. Yin, Y. Shi, and Z. Xue. Intelligent intrusion detection based on federated learning aided long short-term memory. *Physical Communication*, 42:101157, 2020. ISSN 1874-4907. doi: https://doi.org/10.1016/j.phycom.2020.101157. 96, 97

Appendix A

RRIoT Additional Results

A.1 Effects of increasing testing dataset size

We acknowledge the importance of determining the effect of the size of the test dataset. A different train-test split of data can potentially lead to different results. We thus provide a sensitivity experiment that provides our models' performance wherein additional data is allocated to the testing dataset. In our original analyses, we have performed a train-test split of 20%; in this experiment, we provide the results for a train-test split of 2/3 training data and 1/3 testing data. Our results for our models are delineated in Tables A.1, A.2, and A.3.

Table A.1 delineates the results of our models' performance in which there is a 67%-33% train-test split on the TON-IoT GPS telemetry dataset. Interestingly, in this instance when analyzing our models with a greater testing dataset size, we see a slight improvement of performance compared to previous analyses. Notably, we find that when this experiment was ran on our models, our novel RIoT statistically outperformed other models in all performance metrics. Additionally, we also observe a tighter spread of confidence intervals compared to other experiments. While our novel RIoT performed the best, we also observe that our other novel RL models generally outperformed other benchmark ML models.

Table A.1: Results from the base models using the TON-IoT GPS dataset with a larger test dataset size. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	74.29 (0.27)	72.08 (0.32)	79.94 (2.13)	78.22 (0.65)	79.48 (3.45)	82.98 (1.71)	**89.68* (1.53)**
Precision	77.20 (0.05)	81.00 (1.17)	81.81 (2.01)	70.96 (2.48)	83.68 (2.41)	85.43 (0.77)	**89.63* (1.56)**
Recall	74.29 (0.27)	72.08 (0.32)	79.94 (2.13)	78.22 (0.65)	79.48 (3.45)	82.98 (1.71)	**89.68* (1.53)**
F1-Score	74.55 (0.26)	75.53 (0.58)	80.49 (2.07)	73.00 (1.43)	79.31 (3.20)	83.49 (1.35)	**89.22* (1.58)**
G-Mean	85.45 (0.16)	82.11 (0.51)	88.72 (1.25)	86.05 (2.60)	88.46 (2.10)	90.26 (0.71)	**93.68* (1.02)**

Table A.2: Results from the base models using the TON-IoT Modbus dataset with a larger test dataset size. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	68.49 (0.26)	68.49 (0.46)	59.33 (9.07)	**77.20* (2.87)**	10.57 (6.20)	7.99 (3.46)	15.15 (15.39)
Precision	**99.99* (0.00)**	**99.99* (0.00)**	77.56 (21.78)	68.58 (5.26)	51.90 (8.90)	1.03 (0.69)	33.89 (27.49)
Recall	68.49 (0.26)	68.49 (0.46)	59.33 (9.07)	**77.20* (2.87)**	10.57 (6.20)	7.99 (3.46)	15.15 (15.39)
F1-Score	**81.29* (0.19)**	**81.29* (0.32)**	65.78 (15.20)	71.37 (5.26)	11.77 (8.90)	1.40 (0.69)	10.35 (17.85)
G-Mean	**84.25 (0.13)**	**84.25 (0.15)**	59.97 (13.31)	81.60 (12.09)	29.56 (6.44)	25.76 (7.75)	30.06 (12.24)

Table A.3: Results from the base models using the TON-IoT Weather dataset with a larger test dataset size. Statistically significant results are indicated by an asterisk (*).

	Model						
	SVM	**NB**	**MLP**	**DQN**	**AE-RL**	**AE-Dueling DQN**	**RIoT**
Accuracy	75.02 (0.32)	79.08 (0.24)	79.05 (1.30)	78.41 (0.19)	86.35 (2.84)	**88.87* (1.63)**	87.12 (1.53)
Precision	78.40 (0.11)	80.12 (0.29)	81.13 (1.66)	71.49 (0.75)	87.97 (1.51)	**89.73* (1.31)**	87.94 (1.56)
Recall	75.02 (0.32)	79.08 (0.24)	79.05 (1.30)	78.41 (0.19)	86.35 (2.84)	**88.87* (1.63)**	87.12 (1.53)
F1-Score	75.74 (0.22)	79.44 (0.24)	79.54 (1.27)	73.23 (0.37)	86.42 (2.89)	**89.06* (1.49)**	87.27 (1.58)
G-Mean	85.82 (0.18)	88.16 (0.15)	88.21 (0.78)	87.16 (0.90)	92.30 (1.57)	**93.75* (0.94)**	92.52 (1.02)

Table A.2 delineates the results of our models' performance in which there is a 67%-33% train-test split on the TON-IoT Modbus dataset. Compared to the previous experiment, we observe a notable degradation in model performance, especially on our novel RL models. Our best performing RL model for this experiment was our RIoT; however, we see that other benchmark ML models performed significantly better. Our SVM and NB classifiers demonstrated the best precision, F1-score and G-mean on the test dataset, and we observed the best accuracy and recall on the base DQN.

Table A.3 delineates the results of our models' performance in which there is a 67%-33% train-test split on the TON-IoT Weather dataset. Overall, we observe better performance on our novel RL models as compared to our benchmark ML models. Our best performing model was the AE-Dueling DQN model, which statistically outperformed all other models across all performance metrics of accuracy, precision, recall, F1-score, and G-mean. Our RIoT was the second best performing model, and the AE-RL model was the third best performing model.

From our additional experiment on the testing dataset size, we observe overall mixed results from our novel RL algorithms, to include RIoT. Interestingly, we do find that our novel RL models' performance increase when applying the TON-IoT GPS telemetry dataset. However, we observe a drastic decrease in performance for the TON-IoT Modbus telemetry dataset. Our models performed similarly if not slightly better on the TON-IoT Weather dataset. This observation thus suggests that our RL model thus needs more training data in order to better detect threats in the Modbus telemetry data as opposed to the GPS and Weather datasets. Our observations thus suggest utilizing specific algorithms for specific IoT devices. A change in the best performing models was observed in this experiment; Our RIoT was the best on the GPS telemetry dataset, while our AE-Dueling DQN was the best on the Weather telemetry dataset. Our RIoT was the best RL model on the Modbus telemetry dataset but was not the best performing model overall.

A.2 SAGE Plots

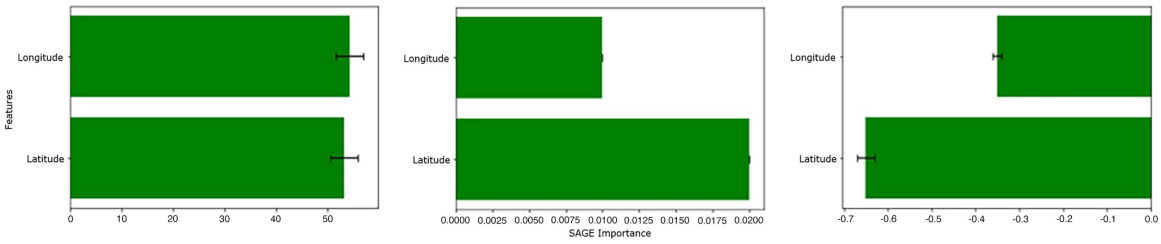


Figure A.1: Analysis I SAGE Plots presented for the GPS telemetry dataset on the AE-RL, AE-Dueling DQN, and RIoT Models, respectively.

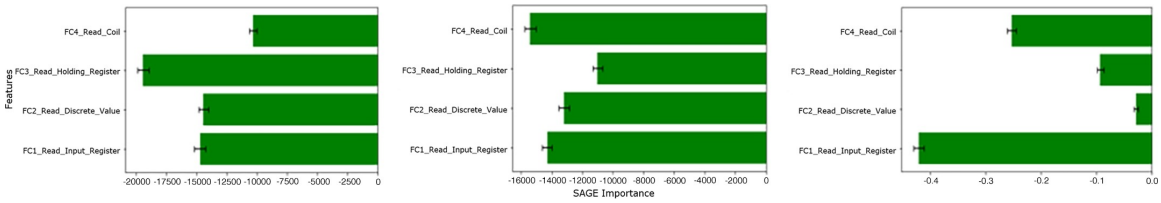


Figure A.2: Analysis I SAGE Plots presented for the Modbus telemetry dataset on the AE-RL, AE-Dueling DQN, and RIoT Models, respectively.

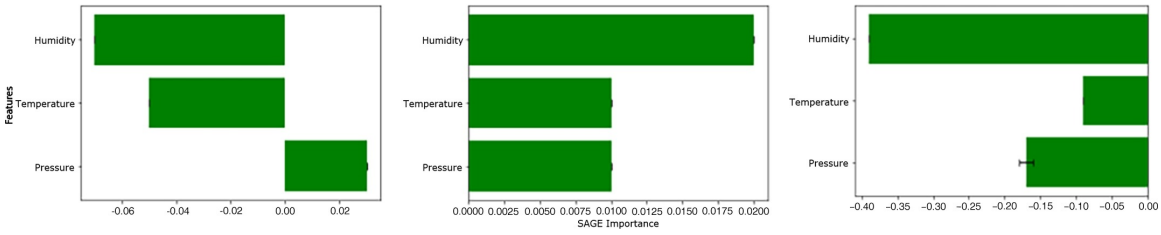


Figure A.3: Analysis I SAGE Plots presented for the weather telemetry dataset on the AE-RL, AE-Dueling DQN, and RIoT Models, respectively.

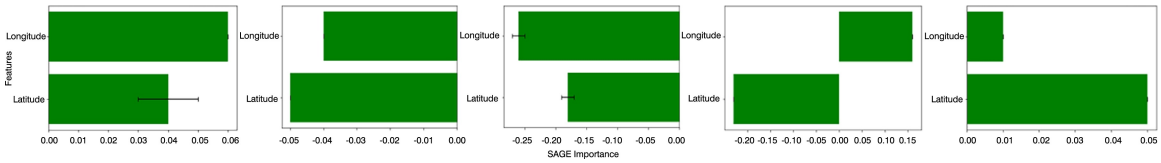


Figure A.4: Analysis II SAGE Plots presented for the GPS telemetry dataset on the AE-DRQN, AE-Dueling DRQN, and RRIoT Models (RRIoT-A, RRIoT-C, and RRIoT-AC), respectively.

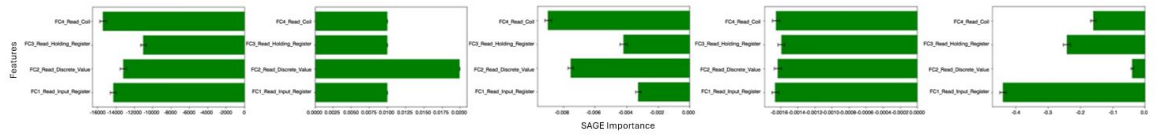


Figure A.5: Analysis II SAGE Plots presented for the Modbus telemetry dataset on the AE-DRQN, AE-Dueling DRQN, and RRIoT Models (RRIoT-A, RRIoT-C, and RRIoT-AC), respectively.

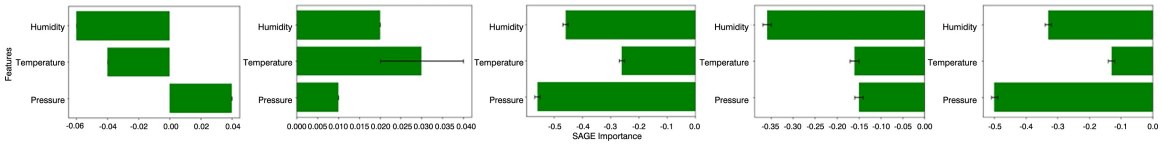


Figure A.6: Analysis II SAGE Plots presented for the weather telemetry dataset on the AE-DRQN, AE-Dueling DRQN, and RRIoT Models (RRIoT-A, RRIoT-C, and RRIoT-AC), respectively.

Appendix B

FLOWS Additional Results

B.1 Normal and zero-day client numerical results

Table B.1 provides a tabular overview of results comparing benchmark federated learning to our FLOWS framework wherein one client has a normal TPM and another client has a modified TPM in which the client is unlikely to transition, $\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$. As observed through the table, we find that our FLOWS framework outperforms the benchmark federated learning model across all metrics. Additionally, we observe that our FLOWS framework statistically outperforms the federated benchmark on all metrics except for G-mean in 25 rounds, G-mean and F1 on 50 rounds, and G-mean, F1, and recall on 75 rounds. Our FLOWS performance thus demonstrates that our novel FLOWS framework reduces the total false positives compared with the federated benchmark.

B.2 Normal and resilient client numerical results

We additionally provide results for comparing benchmark federated learning to our FLOWS framework wherein one client has a normal TPM and another client has a modified TPM in which the client is unlikely to transition out of a normal state,

Table B.1: Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset with one client having a normal TPM and the other client tending to stay in either a normal or attacked state with a TPM $= \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated Benchmark*	0.8669 (0.0005)	0.7948 (0.0045)	0.9568 (0.0026)	0.8645 (0.0044)	0.8711 (0.0540)
	2	*FLOWS Framework*	**0.9207*** **(0.0051)**	**0.9668*** **(0.0261)**	**0.9756*** **(0.0007)**	**0.9017*** **(0.0119)**	**0.9091** **(0.0119)**
50	2	*Federated Benchmark*	0.8865 (0.0030)	0.8436 (0.0452)	0.9309 (0.0452)	0.8795 (0.0300)	0.8847 (0.0465)
	2	*FLOWS Framework*	**0.9136*** **(0.0087)**	**0.9657*** **(0.0288)**	**0.9758*** **(0.0171)**	**0.8849** **(0.0119)**	**0.8943** **(0.0184)**
75	2	*Federated Benchmark*	0.8812 (0.0032)	0.8557 (0.0425)	0.9757 (0.0030)	0.8690 (0.0419)	0.8777 (0.0516)
	2	*FLOWS Framework*	**0.9167*** **(0.0039)**	**0.9651*** **(0.0124)**	**0.9758** **(0.0049)**	**0.8913** **(0.0162)**	**0.8999** **(0.0089)**

$\begin{bmatrix} 0.9 & 0.1 \\ 0.9 & 0.1 \end{bmatrix}$. As observed through the table, we find that our FLOWS framework statistically outperforms the benchmark federated learning model across the recall, precision, F1, and G-mean metrics. We do observe the federated benchmark does statistically outperform our FLOWS framework on the recall metrics; however, our FLOWS model still manages to reduce the incidence of false positives and demonstrates overall better performance. Our results are described in Table B.2.

B.3 Normal and vulnerable client numerical results

Table B.3 provides a tabular overview of results comparing benchmark federated learning to our FLOWS framework wherein one client has a normal TPM and another client has a modified TPM in which the client is unlikely to transition out of an attacked state, $\begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}$. We do observe in this scenario our federated benchmark generally tended to outperform our FLOWS framework on all metrics except for the recall metric. While our model did not outperform as well as the federated benchmark, the spread of results between the federated benchmark and our FLOWS framework overlapped.

Table B.2: Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset with one client having a normal TPM and the other client tending to stay in a normal state with a TPM = $\begin{bmatrix} 0.9 & 0.1 \\ 0.9 & 0.1 \end{bmatrix}$. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated*	0.8116	0.7099	**0.9768***	0.8197	0.8162
		Benchmark	(0.0508)	(0.0702)	**(0.0190)**	(0.0391)	(0.0528)
	2	*FLOWS*	**0.9192***	**0.9056***	0.9103	**0.9069***	**0.9176***
		Framework	**(0.0108)**	**(0.0390)**	(0.0251)	**(0.0096)**	**(0.0077)**
50	2	*Federated*	0.7688	0.6612	**0.9844***	0.7885	0.7686
		Benchmark	(0.0561)	(0.0695)	**(0.0154)**	(0.0417)	(0.0607)
	2	*FLOWS*	**0.9167***	**0.9017***	0.9104	**0.9046***	**0.9153***
		Framework	**(0.0158)**	**(0.0464)**	(0.0253)	**(0.0138)**	**(0.0124)**
75	2	*Federated*	0.7549	0.6454	**0.9868***	0.7784	0.7533
		Benchmark	(0.0498)	(0.0609)	**(0.0130)**	(0.0369)	(0.0540)
	2	*FLOWS*	**0.8915***	**0.8517***	0.9305	**0.8844***	**0.8922***
		Framework	**(0.0508)**	**(0.0940)**	(0.0374)	**(0.0410)**	**(0.0484)**

Table B.3: Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset with one client having a normal TPM and the other client tending to stay in an attacked state with a TPM = $\begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}$. Statistically significant results are indicated by an asterisk (*).

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated*	**0.9258**	0.9680	**0.8570**	**0.9088**	**0.9154**
		Benchmark	**(0.0021)**	(0.0180)	**(0.0169)**	**(0.0031)**	**(0.0038)**
	2	*FLOWS*	0.9222	**0.9712**	0.8451	0.9034	0.9102
		Framework	(0.0058)	**(0.0129)**	(0.0234)	(0.0091)	(0.0092)
50	2	*Federated*	0.9137	**0.9795**	0.8182	**0.8894**	**0.8972**
		Benchmark	(0.0256)	**(0.0195)**	(0.0691)	**(0.0408)**	**(0.0365)**
	2	*FLOWS*	**0.9149**	0.9714	**0.8281**	0.8853	0.8938
		Framework	**(0.0498)**	(0.0136)	**(0.1209)**	(0.1248)	(0.1135)
75	2	*Federated*	**0.9171**	**0.9751**	0.8304	**0.8951**	**0.9026**
		Benchmark	**(0.0215)**	**(0.0220)**	(0.0605)	**(0.0343)**	**(0.0309)**
	2	*FLOWS*	0.9171	0.9694	**0.8348**	0.8913	0.8994
		Framework	(0.0406)	(0.0114)	**(0.0989)**	(0.1019)	(0.0927)

B.4 Effects of varying partial rewards and punishments

Table B.4 provides a tabular overview of results comparing benchmark federated learning to our FLOWS framework with a different reward structure for fully correct classification, partially correct classification, and incorrect classification, respectively. We observe in this analysis that our federated benchmark tends to outperform our FLOWS framework in most reward structure variations except for a reward structure wherein our RL agent obtains a reward of 10 for fully correct, 1 for partially correct, and -10 for incorrect classifications. This variation demonstrated the best performance with respect to accuracy, F1, and G-mean metrics compared to both the federated learning and to other reward structures. Noteworthy is that when the RL reward structure is modified to be 10, 1, -1 for fully correct, partially correct, and incorrect classification, we observe that the precision is higher; our FLOWS framework outputs less false positives. Conversely, we observe when the RL reward structure is modified to be 100, 1, -10 for fully correct, partially correct, and incorrect classification, we observe that the recall is higher; our FLOWS framework outputs less false negatives with a greater incentive to be fully correct.

Table B.4: Results from our benchmark and FLOWS framework models using two clients on our intrusion detection dataset for varying rewards for full correct, partial, and incorrect classification, respectively. Statistically significant results are indicated by an asterisk (*)

Round	Clients	Model	Accuracy	Precision	Recall	F1-Score	G-Mean
25	2	*Federated Benchmark*	0.9225 (0.0133)	0.9287 (0.0473)	0.8942 (0.0341)	0.9094 (0.0116)	0.9180 (0.0110)
	2	*1, 1, -1 FLOW Rewards*	0.8903 (0.0397)	0.8374 (0.0730)	0.9405 (0.0266)	0.8830 (0.0326)	0.8933 (0.0371)
	2	*10, 1, -1 FLOW Rewards*	0.8943 (0.0782)	**0.9746* (0.0141)**	0.7775 (0.1918)	0.8479 (0.1477)	0.8650 (0.1226)
	2	*100, 1, -1 FLOW Rewards*	0.9030 (0.0698)	0.9714 (0.0215)	0.7986 (0.1727)	0.8589 (0.1774)	0.8695 (0.1644)
	2	*1, 1, -10 FLOW Rewards*	0.9142 (0.0342)	0.9617 (0.0200)	0.8361 (0.0920)	0.8906 (0.0576)	0.9003 (0.0504)
	2	*10, 1, -10 FLOW Rewards*	**0.9306* (0.0007)**	0.9567 (0.0051)	0.8790 (0.0064)	**0.9162* (0.0013)**	**0.9233* (0.0016)**
	2	*100, 1, -10 FLOW Rewards*	0.8834 (0.0514)	0.8263 (0.0834)	**0.9449* (0.0282)**	0.8780 (0.0407)	0.8862 (0.0510)
	2	*1, 1, -100 FLOW Rewards*	0.9142 (0.0342)	0.9617 (0.0200)	0.8361 (0.0920)	0.8906 (0.0576)	0.9003 (0.0504)
	2	*10, 1, -100 FLOW Rewards*	0.9165 (0.0041)	0.9067 (0.0254)	0.9006 (0.0206)	0.9030 (0.0030)	0.9143 (0.0022)
	2	*100, 1, -100 FLOW Rewards*	0.8995 (0.0245)	0.8526 (0.0509)	0.9342 (0.0216)	0.8900 (0.0204)	0.9022 (0.0219)
50	2	*Federated Benchmark*	0.9015 (0.0308)	0.8508 (0.0652)	**0.9499 (0.0292)**	0.8951 (0.0263)	0.9044 (0.0283)
	2	*1, 1, -1 FLOW Rewards*	0.8951 (0.0337)	0.8449 (0.0638)	0.9383 (0.0241)	0.8869 (0.0278)	0.8980 (0.0313)
	2	*10, 1, -1 FLOW Rewards*	0.9023 (0.0685)	**0.9755* (0.0142)**	0.7953 (0.1677)	0.8631 (0.1294)	0.8773 (0.1073)
	2	*100, 1, -1 FLOW Rewards*	0.9095 (0.0507)	0.9700 (0.0202)	0.8150 (0.1281)	0.8760 (0.1268)	0.8855 (0.1176)
	2	*1, 1, -10 FLOW Rewards*	0.9148 (0.0341)	0.9586 (0.0198)	0.8407 (0.0921)	0.8918 (0.0573)	0.9016 (0.0503)
	2	*10, 1, -10 FLOW Rewards*	**0.9307* (0.0007)**	0.9572 (0.0050)	0.8786 (0.0062)	**0.9162* (0.0012)**	**0.9232* (0.0015)**
	2	*100, 1, -10 FLOW Rewards*	0.8862 (0.0475)	0.8304 (0.0780)	0.9436 (0.0271)	0.8800 (0.0376)	0.8890 (0.0471)
	2	*1, 1, -100 FLOW Rewards*	0.9148 (0.0341)	0.9586 (0.0198)	0.8407 (0.0921)	0.8918 (0.0573)	0.9016 (0.0503)
	2	*10, 1, -100 FLOW Rewards*	0.9162 (0.0038)	0.9046 (0.0245)	0.9021 (0.0202)	0.9028 (0.0028)	0.9142 (0.0021)
	2	*100, 1, -100 FLOW Rewards*	0.9009 (0.0230)	0.8557 (0.0472)	0.9325 (0.0199)	0.8911 (0.0192)	0.9035 (0.0205)
75	2	*Federated Benchmark*	0.8981 (0.0054)	0.9216 (0.0259)	0.8549 (0.0336)	0.8859 (0.0047)	0.8987 (0.0037)
	2	*1, 1, -1 FLOW Rewards*	0.9009 (0.0287)	0.8562 (0.0544)	0.9339 (0.0206)	0.8917 (0.0236)	0.9033 (0.0266)
	2	*10, 1, -1 FLOW Rewards*	0.9048 (0.0648)	**0.9760* (0.0151)**	0.8009 (0.1588)	0.8679 (0.1224)	0.8813 (0.1015)
	2	*100, 1, -1 FLOW Rewards*	0.8929 (0.0867)	0.9721 (0.0210)	0.7744 (0.2130)	0.8340 (0.2217)	0.8465 (0.2055)
	2	*1, 1, -10 FLOW Rewards*	0.9184 (0.0282)	0.9571 (0.0162)	0.8502 (0.0762)	0.8978 (0.0474)	0.9069 (0.0416)
	2	*10, 1, -10 FLOW Rewards*	**0.9308* (0.0006)**	0.9573 (0.0040)	0.8788 (0.0050)	**0.9163* (0.0010)**	**0.9234* (0.0012)**
	2	*100, 1, -10 FLOW Rewards*	0.8948 (0.0406)	0.8462 (0.0674)	**0.9374* (0.0238)**	0.8870 (0.0322)	0.8972 (0.0400)
	2	*1, 1, -100 FLOW Rewards*	0.9184 (0.0282)	0.9571 (0.0162)	0.8502 (0.0762)	0.8978 (0.0474)	0.9069 (0.0416)
	2	*10, 1, -100 FLOW Rewards*	0.9157 (0.0032)	0.9013 (0.0205)	0.9044 (0.0168)	0.9025 (0.0023)	0.9141 (0.0017)
	2	*100, 1, -100 FLOW Rewards*	0.9046 (0.0194)	0.8630 (0.0398)	0.9300 (0.0166)	0.8943 (0.0162)	0.9068 (0.0174)

Vita

Curtis Rookard obtained his B.A. in Mathematics from Thomas Edison State University in 2018 and proceeded to earn his M.S. in Biostatistics from Middle Tennessee State University in 2020. Prior to beginning his PhD education at the University of Tennessee in Industrial and Systems Engineering, he worked as a statistical programmer and assisted statisticians with clinical trials. He became interested with the cybersecurity field during his military service and has published works on the topic of machine learning-assisted intrusion detection systems with his advisor, Dr. Anahita Khojandi. For his PhD studies, he was awarded the prestigious Department of Defense Science, Mathematics, and Research for Transformation (SMART) scholarship. He is a student member of the IEEE and the ASA.