12-8-2023

# Explainable Intrusion Detection Systems using white box techniques

Jesse Ables
*Mississippi State University*, jha92@msstate.edu

# Explainable Intrusion Detection Systems

# using white box techniques

By

Jesse Ables

Approved by:

Sudip Mittal (Major Professor)
Shahram Rahimi
Ioana Banicescu
Stefano Iannucci
Maxwell Young
T.J. Jankun-Kelly (Graduate Coordinator)
Jason M. Keith (Dean, College of Engineering)

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctorate of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2023

Name: Jesse Ables

Date of Degree: December 8, 2023

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Sudip Mittal

Title of Study: Explainable Intrusion Detection Systems using white box techniques

Pages of Study: 119

Candidate for Degree of Doctorate of Science

Artificial Intelligence (AI) has found increasing application in various domains, revolutionizing problem-solving and data analysis. However, in decision-sensitive areas like Intrusion Detection Systems (IDS), trust and reliability are vital, posing challenges for traditional black box AI systems. These black box IDS, while accurate, lack transparency, making it difficult to understand the reasons behind their decisions. This dissertation explores the concept of eXplainable Intrusion Detection Systems (X-IDS), addressing the issue of trust in X-IDS. It explores the limitations of common black box IDS and the complexities of explainability methods, leading to the fundamental question of trusting explanations generated by black box explainer modules.

To address these challenges, this dissertation presents the concept of white box explanations, which are innately explainable. While white box algorithms are typically simpler and more interpretable, they often sacrifice accuracy. However, this work utilized white box Competitive Learning (CL), which can achieve competitive accuracy in comparison to black box IDS. We introduce Rule Extraction (RE) as another white box technique that can be applied to explain black

box IDS. It involves training decision trees on the inputs, weights, and outputs of black box models, resulting in human-readable rulesets that serve as global model explanations. These white box techniques offer the benefits of accuracy and trustworthiness, which are challenging to achieve simultaneously.

This work aims to address gaps in the existing literature, including the need for highly accurate white box IDS, a methodology for understanding explanations, small testing datasets, and comparisons between white box and black box models. To achieve these goals, the study employs CL and eclectic RE algorithms. CL models offer innate explainability and high accuracy in IDS applications, while eclectic RE enhances trustworthiness. The contributions of this dissertation include a novel X-IDS architecture featuring Self-Organizing Map (SOM) models that adhere to DARPA's guidelines for explainable systems, an extended X-IDS architecture incorporating three CL-based algorithms, and a hybrid X-IDS architecture combining a Deep Neural Network (DNN) predictor with a white box eclectic RE explainer. These architectures create more explainable, trustworthy, and accurate X-IDS systems, paving the way for enhanced AI solutions in decision-sensitive domains.

Key words: Intrusion Detection, Artificial Intelligence, Explainable Artificial Intelligence, Explainabile Intrusion Detection Systems, Competitive Learning, Rule Extraction

DEDICATION

*To my cats Mipha and Poseidon*

ACKNOWLEDGEMENTS

The journey through my Ph.D. has been a long and difficult process. This process was made much easier by having great mentors and colleagues. I would like to begin by thanking my advisor Sudip Mittal for leading me on this journey. His deep knowledge of academia was a major factor in my development. My ability to write and present has improved significantly since meeting him. He has helped foster my passion for research and helped me immensely in creating this dissertation.

I would also like to thank Stefano Iannucci who caused me to begin my Ph.D. studies. He saw potential in me as a student and offered to be my mentor. Without Stefano's assistance, I would not have begun my journey to becoming a Ph.D. His impressive background in research and industry gave me one of the best foundations one could want for a successful Ph.D. career.

I would like to thank the rest of my committee members: Shahram Rahimi, Ioana Banicescu, and Maxwell Young for their guidance and advice on this dissertation. Shahram and Ioana have both provided critical feedback for this and all of my work. Their collective knowledge and experience have made me a much stronger independent researcher. Max has also helped me to look at my work outside the lens of cyber security. He has posed many insightful questions that have helped improve my research.

I would like to thank the US Army Engineer Research and Development Center (ERDC), Dr. Maria Seale, and other hard-working individuals at ERDC for their support of this work.

Most importantly, I'm thankful to my family and friends for their unending support. My parents, brother, and sister-in-law have all been supportive of my entire academic career. Without their help and support throughout the years, it is doubtful I would have made it this far. My friends online and off have been a great help to my mental fortitude. Lastly, I'm thankful to my cats Mipha and Poseidon for their love and comfort.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

Artificial Intelligence (AI) has become more prevalent in our everyday lives. Increasingly, academia, industry, and government leverage AI for problem-solving and data analysis [42]. We have seen AI used to automate the artistic process [82], operate self-driving cars [61], make medical diagnoses [43], and protect networking infrastructure [46]. For decision-insensitive domains, properties such as privacy, trust, or reliability are not necessary requirements. However, for decision-sensitive domains, such as Intrusion Detection Systems (IDS), these are critical aspects of their functionality. Currently, the most popular approaches for IDS use black box algorithms. These black box algorithms are primarily selected due to their high accuracy and generalizability [103]. However, do we understand why a black box IDS makes a decision? Can we trust black box IDS to make decision-sensitive predictions? The state-of-the-art approaches for IDS, as well as machine learning as a whole, focus on model performance through the lens of model accuracy. This focus on model accuracy has driven the development further away from modeling approaches that are transparent or have a clear notion of explainability. This creates a separation between model inference and the *understanding* of model inference, which in turn gives the inability to confirm model fairness, privacy, reliability, causality, and ultimately trust. This set of problems can be solved by eXplainable Intrusion Detection System (X-IDS).

Currently, black box IDS have techniques that can be used to explain predictions. Modern X-IDS use ubiquitous solutions such as Local Interpretable Model-Agnostic Explanations (LIME) [85] or SHapley Additive exPlanations (SHAP) [60] that can be used out-of-the-box to generate local or global feature importance charts. These explanations can be used to determine which features of an intrusion detection dataset are the most impactful for predictions. However, these solutions are also black box. To create explanations, these explainer modules create black box surrogates that approximate the original model. This leads to the question, “If I cannot trust predictions from a black box AI, how can I trust explanations from a black box explainer?"

One potential solution to these explainability and trustability problems is white box explanation techniques. White box algorithms are innately explainable which leads to more trustworthy predictions. As their name implies, these algorithms are transparent. Generally, white box algorithms are simple to understand and repeatable by humans. They can be represented by explainable visualizations or human-readable rulesets. This leads to users being able to understand how the model works and trust the predictions they make. However, the benefit they provide with simplicity causes issues with accuracy. These algorithms are typically not very accurate with complex datasets. Yet, there is a white box family of algorithms that is able to be highly accurate when compared to black box IDS.

Competitive Learning (CL) is a family of algorithms that uses competition to train neurons, nodes, or best-fit lines. During training, these algorithms use a competitive metric such as distance or similarity to choose winner nodes. The winner node, also known as the Best Matching Unit (BMU), has its and its neighbor’s weights adjusted closer to the training sample. This gives those nodes an advantage when similar training data is used, effectively creating a feature detector [48].

The training process continues until nodes begin to cluster together forming separations between different kinds of data.

Rule Extraction (RE) is also a white box technique that can be used to explain black box IDS. RE, as its name implies, extracts rules from a black box system and generates human-readable rulesets that act as global model explanations. This is done by training one or more decision trees on the black box model's inputs, weights, and outputs. Each branch in the decision tree can be extracted to form a rule. The rules are then combined to create an explainable ruleset. Using this method alongside black box models allows the user to utilize the accuracy of black box AI and the trustability of white box models.

These white box techniques can be effective for creating X-IDS. They can be used to address gaps in the current literature. First, many white box IDS are often not very accurate, especially when compared to their black box counterpart. Second, there is little methodology used to understand or use explanations generated from an explainable system. Third, many of these white box algorithms are only trained and tested on small datasets both in the number of samples and features. Fourth, there is little comparison in the literature between white box and black box models. Lastly, rule extraction explainers have limited use in deep neural network IDS. The goals of this dissertation seek to remedy these gaps. We demonstrate that there is a class of white box algorithms that can have competitive accuracy when compared to black box algorithms. We create a methodology for understanding and using explanations generated from these algorithms. We train and test our models against common IDS datasets found in the literature. These include NSL-KDD, CIC-IDS-2017, and UNSW-NB15. Our models are compared to black box models in both explainability and accuracy. Lastly, we design a rule extraction algorithm that can be run on large IDS datasets.

This dissertation chooses to use Competitive Learning (CL) and eclectic Rule Extraction (RE) to achieve these goals. The innate explainability of CL algorithms allows them to create visual and statistical explanations that users can use to understand the model's reasoning. Their ability to compress high-dimensional datasets into a 2D representation allows them to be highly accurate on IDS datasets. Additionally, an eclectic RE algorithm is used to extract rules from a DNN IDS. These models and techniques can be used solely as X-IDS systems or as companion explainers for accurate black box systems. In this dissertation, the chosen approach involves creating X-IDS using CL algorithms to build a more explainable and trustworthy IDS. Additionally, since black box AI is a popular technique, a hybrid X-IDS is created using a Deep Neural Network (DNN) predictor and a white box RE explainer. The contributions of this dissertation are:

- A novel X-IDS architecture featuring a SOM, built using DARPA's proposed guidelines for an explainable system. This system is able to produce robust, explanatory visualizations of the SOM model and create accurate IDS predictions. We dictate a methodology for understanding and using explanations to further increase the accuracy of the X-IDS architecture.
- An extended X-IDS architecture featuring three CL-based algorithms, built using DARPA's guidelines for an explainable system. Self-Organizing Map, Growing Self Organizing Map, and Growing Hierarchical Self Organizing Map models are used to create explanatory visualizations and accurate predictions. The innately explainable CL models have similar accuracy and are more trustworthy when compared to error-based learning models.
- A hybrid X-IDS architecture that uses a DNN predictor and a white box, eclectic RE explainer. Using white box RE, the X-IDS is able to create more trustworthy explanations than black box surrogate explainers. Additionally, experiments are run to demonstrate the customizability of the eclectic RE algorithm so that it can be run on large IDS datasets.

## 1.2 Overview

This dissertation is divided into six chapters. Chapter 2 discusses the background, related works, and algorithms used in this dissertation. This section covers important information regarding explainable AI, intrusion detection, explainable intrusion detection, competitive learning, and rule

extraction. Chapter 3 overviews the datasets used in this work. This dissertation uses three datasets: NSL-KDD, CIC-IDS-2017, and UNSW-NB15. Their historical and statistical characteristics are discussed as well as an overview of the attacks each dataset contains.

Chapter 4 is the first contribution of this dissertation. In this work, an X-IDS architecture is created that uses Self Organizing Maps to make predictions and explanations. The architecture is based on DARPA's recommendations for explainable systems. The X-IDS is able to create visual and statistical explanations that users can use to understand how and why the model makes predictions. The experiment and evaluation for this work involve discussing the architecture's explainability. This includes how well the model clustered malicious and benign data and methods for using explanations to understand the model. Additionally, traditional accuracy experiments are run to demonstrate the model's effectiveness compared to other black box models.

Chapter 5 extends the previous X-IDS architecture to include two more CL algorithms: Growing Self Organizing Maps (GSOM) and Growing Hierarchical Self Organizing Maps (GHSOM). This work compares CL algorithms to their black box counterpart Error Based Learning (EBL) algorithms. Similar to the previous work, the X-IDS architecture is based on DARPA's recommendation for explainable systems. An explainability analysis is conducted to discuss how explanations can be used to understand the model. Explanations include both visual and statistical variants. Lastly, a traditional performance analysis is run that allows the architecture to be compared to EBL models found in the literature.

Chapter 6 seeks to bridge the gap in this work between black box AI and white box explainer techniques. This work chooses to use an eclectic RE algorithm that can be used to generate explanations for a DNN IDS. Explanations are given in the form of rulesets. Rulesets can be parsed

or searched to get a global understanding of the model, and rules can be returned with samples for a local explanation. A set of experiments is run demonstrating the RE algorithm's accuracy and customizability. Additionally, the explainability of this method is discussed with regard to the experiments.

Lastly, Chapter 7 concludes this work. It gives a brief summary of this dissertation's novel contributions. Additionally, potential future works are described.

# CHAPTER 2

# BACKGROUND AND RELATED WORKS

Chapter 2 details the relevant background knowledge needed to understand later chapters in this dissertation. In Section 2.1, intrusion detection systems are discussed along with the current state of the art and philosophy. Section 2.2 contains background information for explainable artificial intelligence. Section 2.3 briefly overviews explainable intrusion detection and its various techniques. Section 2.4 then details the various competitive learning techniques followed by a description of rule extraction in section 2.5.

## 2.1 Intrusion Detection Systems

An *intrusion* refers to an action that obtains unauthorized access to a network or system [24]. An Intrusion Detection System (IDS) consists of tools, methods, and resources that help a Cyber Security Operation Center (CSoC) protect an organization by detecting an intrusion [11, 65]. IDS can be categorized into operation-based classes, such as signature, anomaly, and hybrid. Signature-based IDS operate by preventing known attacks from accessing a network. The IDS compares incoming network traffic to a database of known attack signatures. Notably, this method has difficulty in preventing *zero-day* attacks [91]. Anomaly-based IDSs look for patterns in incoming traffic to recognize potential threats and leverage complex AI models [19, 64, 71]. A significant drawback of this approach is the tendency for such systems to categorize legitimate, unseen behavior

as anomalous. Hybrid-based IDS incorporates the design philosophy of both signature-based and anomaly-based IDS to improve the detection rate while minimizing false positives [79, 95].

Current work on AI enabled anomaly-based IDS can be further divided into black box and white box models [71]. White box models are considered *easy to understand* by an expert. This allows the expert to analyze the decision process and understand how the model renders its decision. This (semi-) transparent property allows white box models to be deployed in decision sensitive domains, where auditing the decision process is a requirement. White box models may use regression-based approaches [94], decision trees [62], and Self Organizing Maps (SOMs) [53]. Black box models, on the other hand, have an opaque decision process. This opaqueness property makes establishing the relationship between inputs and the decision difficult, if not outright impossible. Black box models comprise nearly all the AI enabled state-of-the-art approaches for IDS, as the focus is traditionally on model performance, not explainability. Examples of popular black box model techniques are Isolation Forest [58], One-Class SVM [89], and Neural Networks [108].

## 2.2 Explainable Artificial Intelligence

The notion of an Explainable Artificial Intelligence system (XAI) dates back to the 1970s. Moore et al. [67] surveyed works from the 1970s to the 1980s, detailing early methods of explanations. Some early explanations consisted of canned text and code translations, such as the 1974 explainer MYCIN [92]. We can find a more current definition of XAI by the Defense Advanced Research Projects Agency (DARPA) [28]. DARPA defines XAI as 'systems that are able to explain their reasoning to a human user, characterize their strengths and weaknesses, and convey a sense of their future behavior'. An XAI system that follows this definition offers some form of

justification for its action, leading to more trust and understanding of the system. The explanations from an XAI system help the user not only in using and maintaining the AI model but also helping users complete tasks in parallel with the AI system. Tasks can include doctors making medical decisions [37, 56, 92], credit score decisions [21], detecting counterfeit banknotes [36], advance maintenance [73], or CSoC operators defending a network [4, 22, 28].

The current literature consists of many different black box models being used alongside explanation techniques. Common explainer modules for black box models are Local Interpretable Model-agnostic Explanations (LIME) [85], SHapely Additive exPlantions (SHAP) [60], and Layer-wise Relevance Propagation (LRP) [14]. Modern techniques for explaining black box models consist of creating surrogate models that generate explanations either locally or globally. Other methods involve propagating predictions backward in a neural network or decomposing a gradient. More novel approaches have also experimented with making datasets explainable [40] or making graphical user interfaces for explainable systems [102].

### 2.3 Explainable Intrusion Detection Systems

Explainable Intrusion Detection Systems (X-IDS) are still an emerging sub-genre. The need for explainability in IDS is becoming increasingly necessary. In decision sensitive domains, black boxes obfuscate the decision making process causing a lack of trust in predictions. The users need to be confident in the predictions or recommendations computed by an IDS. Understandable and trustworthy explanations allow users to perform their tasks correctly. The stakeholders of an IDS (e.g. CSoC operators, developers, and investors) are individuals who will be dependent on the performance of the system [71]. CSoC operators will be performing defense actions based on

prediction and explanation results. Developers can use explanations to fortify the model in areas where it is weak. Investors may need explanations to help them make their company's budgeting decisions.

There are many examples of X-IDS being used in research today. A survey by Neupane et al. [71] describes in detail different X-IDS systems. Many black box implementations have been shown using libraries such as SHAP, LIME, or LRP [8,45,100]. There have also been more original explanation frameworks, such as one that involves using the CIA triad to generate explanations [40].

White box models have also been used to create strong X-IDS architectures. The authors of [51] created a regression-based X-IDS based on Hardware Performance Counters (HPC). This work focuses on two types of attacks: microarchitecture and malware. Ridge Regression is used to generate explainable results. A limitation of this work is the use of small datasets. Their model is only trained using four features. Work by [62] and [25] create rule-based X-IDS using Decision Trees (DT). The former work utilized Iterative Dichotomiser 3 (ID3) algorithm because of its ability to mimic the human decision process. The latter work uses expert justification to generate explainable results. These works are limited by their use of small datasets and the small knowledge base generated by experts.

There are a few notable works for white box clustering algorithms. These works utilize the SOM as a means of creating accurate models and trustworthy explanations. Langin et. al. [53] created an algorithm called ANNaBell Island. This work seeks to expand the explainability of the SOM's output. SOM outputs are used to create 3D color-separated hexagonal landscapes. This allows the user to distinguish between different types of attacks and classifications. Notably, this work's main focus is on the explainability of the SOM and not the accuracy. They use a small

dataset to demonstrate the usefulness of their algorithm. Wickramasinghe et. al. [101] use various visualization techniques to make SOMs more explainable. They utilize the U-matrix, t-SNE, histograms, and heatmaps to create visual and statistical explanations. This work primarily focuses on the explainability of the SOM and not on its accuracy.

## 2.4 Competitive Learning

In this section, the theoretical and practical aspects of Competitive Learning (CL) algorithms are described. CL covers a range of algorithms wherein parts of the model compete against one another to represent one aspect of a dataset. This is opposed to a different method of training models known as Error Based Learning (EBL). In the following section, CL is discuss and how it compares to EBL. Current CL based intrusion detection systems are also discussed. Lastly, various CL algorithms that are used later in the dissertation are explained.

### 2.4.1 Error Based Learning vs. Competitive Learning

Neural network training algorithms can be divided into a few categories. One of the most popular categories is Error Based Learning (EBL). The core principle behind EBL is optimization. EBL models are trained through a process known as Empirical Risk Minimization (ERM). Through this process, the Machine Learning (ML) algorithm works to minimize a parameter known as 'loss', which is a metric that measures how poorly a model predicted a specific sample. If the model is correct, loss is given a value of 0, otherwise, loss will be a value greater than 0. Common loss functions include Binary Cross Entropy, Mean Absolute Error, and Poisson. To make use of the loss function, ML algorithms employ an ERM technique. Gradient Descent (GD) is one of the most well known techniques for this purpose. GD works by calculating the slope or gradient at

a given point of a loss function. Normally, this strategy is applied to convex functions, but ML applications are rarely so orderly. After calculating the gradient, GD then takes a *step* down the slope. A *step* can be done for every training sample or a batch of training samples. This changes the weights and biases of a neural network in an effort to lower the loss. This process repeats until the algorithm has converged as close as possible to 0.

The next set of algorithms that can be used is Competitive Learning (CL). CL consists of unsupervised algorithms where nodes *compete* with one another over the right to activate for input data. There are also a few variants of these algorithms that use probabilistic methods rather than neural networks. CL algorithms follow three tenets [86]: (i) all units are the same at the start except for their randomly selected weights, (ii) the 'strength' of each unit is limited, and (iii) units compete to represent a sub-set or 'cluster' of the input data. Using these tenets, nodes in a CL algorithm can represent abstract patterns or features in data. Nodes compete by being closer to the input data. Generally, this is calculated through euclidean distance. The randomly selected weights are then adjusted to be closer to the input data. There are a few common algorithms that implement this tactic: SOMs, K-Means Clustering, and Expectation-Maximization (EM) mixture modeling [88].

One can already begin to see the difference between CL and EBL. Neurons in EBL algorithms represent an activation function rather than mimicking input data. EBLs train towards the goal of minimizing loss from these activation functions. On the other hand, the nodes of CL algorithms contain a vector that is similar to the input data. Training these nodes allow these algorithms to slowly converge toward the inputted samples. Another major difference between these two learning styles is their supervised/unsupervised nature. Many EBLs require a supervised based

learning style such that their loss function can be calculated. However, CL algorithms are able to be trained in an unsupervised manner. Data labels are not needed during the training process. Another advantage for CL based methods is that they tend to be innately explainable. Since the model works to represent clusters in training data, the model can be data-mined for various visual and statistical explanations. Since both the model and explanations are generated in a white box manner, the explanations can be seen as more trustworthy. The same cannot be said for many EBL methods. As mentioned in a previous section, frameworks such as SHAP [60] or LIME [85] may be required to make EBL neural networks explainable. These explanation frameworks are, themselves, black box. Therefore, the explanations they generate may not be as trustworthy as white box alternatives. Lastly, the two algorithm sets predict data differently. EBL algorithms predict data using a loss threshold that causes neuron activation while CL predicts based on proximity.

Generating explanations for EBL and CL algorithms also differ. EBL algorithms are categorically known as black box algorithms. It is difficult to discern what process the algorithm took to create predictions. To remedy this, one can use a surrogate model method such as Local Interpretable Model-agnostic Explanations (LIME) [85], SHapely Additive exPlantions (SHAP) [60], and Layer-wise Relevance Propagation (LRP) [14]. These surrogate models create explanations generally through processes such as perturbation or probabilistic set theory. There are two major problems with using these approaches. First, the use of these algorithms is effectively using a black box to explain a black box. The process for generating the explanations can be difficult to understand, so it may be more difficult for users to trust the explanations. Secondly, surrogate generators can be computationally expensive. Not only does one need to train a model, they must then train a surrogate model afterwards. CL algorithms remedy these problems. Since the algorithms are

already white box, they can easily be explained. Users can create their own custom explanations that they can trust. Additionally, the computational complexity is generally limited by the size of the CL algorithm's map of nodes.

### 2.4.2 Competitive Learning used in Intrusion Detection

In the past, CL algorithms have been used to create many IDS. These studies focused on building accurate IDS and did not discuss explainability. Among these approaches, SOMs were used to create both host-based [55] and network-based [23, 76, 84] IDS. The majority of these methods simply trained a SOM based IDS and illustrated mappings between data points and the associated Best Matching Unit (BMU). The approaches described in [6, 84] use multiple SOMs in conjunction with one another to create a more effective IDS. Only one approach [23] discussed the false positive rate and accuracy of a SOM-based IDS. Their method for prediction involved assigning a label to BMUs based on the training dataset. Using this approach meant that not all SOM units were assigned a label. The authors utilized Gaussian Mixture Modeling (GMM) to make predictions when a testing sample was similar to an unlabeled unit. In our previous work [4], we created an X-IDS architecture based on DARPA's recommended architecture. One of its main features is having user input for correcting or modifying the model or its explanations. Using this architecture, we were able to achieve an accuracy of 91% on NSL-KDD and 80% on CIC-IDS-2017.

In addition, we can look at instances of GSOM-based IDS. A multi-agent GSOM proposed by Palomo et al. [78] was created with the goal of being more accurate on datasets with many different attack types. The Growing SOM should be able to continuously grow as it discovers new attack

types. Their IDS was able to achieve a 90% accuracy and a 1% false positive rate on the KDD CUP 1999 dataset using 38 different attacks. A novel GSOM algorithm was developed in [81] and called Statistics-Enhanced Direct Batch Growth Self-Organizing Map (SE-DBGSOM). One of the goals of using this updated algorithm is to improve the efficiency of inserting new nodes. The authors note that their algorithm improves upon previous GSOMs by reducing the number of 'unnecessary' nodes. This improves both runtime and false positive rates. SE-DBGSOM was able to achieve a greater than 99% accuracy on KDD99 and CICIDS2017 datasets with false positive rates as low as .6%.

GHSOMs have also made an impact in the field of IDS. One inspiring work that created a GHSOM IDS is from the authors Ippoliti et al. [39]. They create an Adaptive GHSOM (A-GHSOM) that uses dynamic normalization scaling, an adaptive growth thresholds, and confidence filtering for reducing inconsistent predictions. We can find other works that make other modifications like adding new metrics for numeric and symbolic data [77], enhancing map initialization and weight distribution [87], and changing growing conditions [105]. Many of these implementations were testing using KDD CUP 1999 or NSL-KDD to great effect.

The final two methods for CL algorithms, EM mixture modeling and K-means clustering, have also been used to create effective intrusion detection systems. Both Bahrolo et al. [12] and Hammad et al. [35] have created EM mixture model IDS that attempt to categorize the different attacks in IDS datasets. Another work uses a combination of decision trees and EM mixture modeling to create an effective IDS with an accuracy of 94.2% on the NSL-KDD dataset [15]. There are a few notable works that use K-means clustering or an ensemble featuring K-means to categorize or predict anomalies. Two methods combine K-means with a Naive Bayes classifier to achieve

high detection rates on KDD'99 and ISCX 2012 [70, 96]. In Li et al. [54], their IDS using solely K-means clustering records a detection rate of 82% on the KDD'99 dataset.

Part of this work focuses on the SOM family of CL algorithms. These innately explainable algorithms have been shown in previous works to be highly accurate for intrusion detection. Their *simple-to-understand* nature is conducive to obtaining great explainable algorithms. In addition, the weights generated by the SOM algorithms are easily visualized for explanations. In the following sections, we describe in detail how each of the SOM algorithms operates.

### 2.4.3 Self-Organizing Maps

Self Organizing Maps (SOMs), sometimes referred to as Kohonen Maps [47, 74], Kohonen Self Organizing Maps [29], or Kohonen Networks [50], are a class of unsupervised machine learning algorithms. SOMs are comprised of a network of individual nodes, each of which has a feature vector of the same size as the dimension of training data. Some implementations also include a *(x,y)* coordinate to allow node movement in a two-dimensional (2D) space. This 2D space is typically represented as a square or a hexagonal grid, to easily visualize the represented space.

POPSOM, outlined in Algorithm 1, is the SOM algorithm chosen for this work [106]. It takes four inputs: the number of rows (n), the number of columns (m), the learning rate (LR), and the total number of epochs (T). Radius is also a common parameter that needs to be set in most SOM algorithms, however, POPSOM calculates its initial radius using $n$, $m$, and LR. The $n$ and $m$ determine the size of the map, while $LR$ is how aggressive the model adjusts its weights. The SOM trains for a total of $T$ epochs before finishing. The algorithm begins by selecting a random training sample. Then, the Best Matching Unit (BMU) is calculated by finding the smallest euclidean

**Algorithm 1** POPSOM Algorithm

**Input:** Rows (n), Columns (m), Learning Rate (LR), Total Epochs (T)
**Output:** Weights (W)

```
BEGIN
Allocate n * m element array W
for each node in W do
    Allocate N element array with random values [0,1]
end for
for Each Training Epoch in T do
    Pick a training sample
    Find Best Matching Unit using Euclidean Distance
    Update BMU elements: w_i = w_i − λ ∗ (w_i − i_i)
    Update BMU Neighbors
    Update Learning Rate
end for
return W
END
```

distance from the training sample to a SOM node. After the BMU is found, it and its neighbors are updated using the formula $w_i = w_i - \lambda * (w_i - i_i)$, where $w$ is the set of BMU weights and $i$ is the set of feature values. $\lambda$ is the learning rate function that considers the current training iteration, the chosen $LR$, and the distance from the BMU. Lastly, the learning rate, neighborhood radius, and current iteration numbers are updated. The function $\lambda$ works in a way that it decreases during the course of the training process.

SOMs have some unique advantages that come with their application. The first is algorithmic simplicity. As shown in Algorithm 1, the brevity of the algorithm helps to maintain the desired properties of algorithmic decomposability and tractability. Additionally, due to its unsupervised nature, SOMs can work on a variety of datasets and applications (e.g. data mining and discovery), not just prediction [75]. By design, SOMs convert high-dimensional data into a lower dimensional representation. This representation can be topologically clustered and explained through visual-

izations [76]. One challenge that comes with the application of SOMs is the selection of the size parameters, as the size does not dynamically adjust and there is no *best size* heuristic [16]. Finally, another challenge with SOMs is their scalability, both in their time complexity, $O(N^2)$, and space complexity. More methods, such as those in [59], are needed to address these challenges.

### 2.4.4 Growing Self-Organizing Maps

SOMs were further improved by dynamically growing the 2D represented space. The Growing Self-Orginaizing Maps (GSOM) was created by Bernd Fritzke in his impactful work [27]. Their work kept the square architecture common to SOMs, but allowed it to grow by adding columns or rows dynamically. Future implementations would implement systems that allowed the SOM to grow node-by-node rather than with full rows or columns [5]. The training process of the GSOM is very similar to that of the SOM other than the growing process.

The GSOM algorithm chosen for this paper is the Direct Batch Growing Self-Organizing Map (DBGSOM) [99]. Its psuedocode can be found in Alg. 2. It takes three inputs: the dataset's dimensions (D), Spread Factor (SF), and Learning Rate (LR). $D$ is the number of features a dataset has. $SF$ determines how quickly new nodes are generated. $LR$ is the same as in the SOM. Another important variable that is not selected by the user is the Cumulative Error (CE). Each node in the GSOM has a $CE$ value. $CE$ is the sum of all the differences between a sample and its BMU. This value slowly accumulates over the course of training.

DBGSOM follows similar tenets as the original GSOM algorithm. The main difference is that it generates new neighboring nodes in a batch process. It is initialized with four starter nodes with randomized weights between 0 and 1. A growth threshold is calculated based on $SF$ which

**Algorithm 2** DBGSOM Algorithm

**Input:** Data Dimension (D), Spread Factor (SF), Learning Rate (LR), Total Epochs (T)
**Output:** Weights (W)

```
BEGIN
Initialization
Initialize 4 starter nodes with random Weights W [0,1]
Calculate Growth Threshold (GT): GT = −D ∗ ln(SF)
Growing Phase
for Each Training Epoch in T do
    Reset Cumulative Error (CE) for all nodes to 0
    Present training samples
    Determine BMU using Euclidean Distance
    Update BMU and Neighboring weights
    Calculate CE for all BMUs
    for all non-boundary nodes do
        Distribute CE to neighbors
    end for
    for all boundary nodes CE > GT do
        Grow depending on number of available neighbor positions
    end for
end for
return W
END
```

is static throughout the training process. After the DBGSOM is initialized, it enters the *Growing Phase*. All nodes have their *CE* reset to 0. Training the GSOM is now similar to training a SOM. Each training sample is presented to the map, and its respective BMU is found. The BMU has its weights and *CE* updated based on the training sample. Additionally, all neighbors of the BMU have their weights updated. After all of the training data has been used to update weights, we find all non-boundary nodes. For each of these nodes, we distribute their *CE* to their neighbors. Lastly, all boundary nodes for which $CE_i > GT$ have a new neighbor node generated next to it.

A major advantage of using this algorithm is the undefined size of the map. SOMs are limited in the fact that they use an unchanging number of nodes. If a map is too small, then different labels from the dataset can begin to merge or take over one another. On the other hand, a larger map may lead to many useless nodes taking up processing time. GSOMs solve this issue by adding new nodes as needed. When the dataset processes a new idea (or a new attack in the case of an IDS dataset), a new set of nodes can be generated with similar weights.

### 2.4.5 Growing Hierarchical Self-Organizing Maps

The SOM field would enter another Renaissance in the form of the Growing Hierarchical Self-Organizing Map (GHSOM). The authors, Dittenbach et al., changed the growing algorithm to not only grow horizontally but also vertically (i.e. hierarchically) [26]. Each layer of the GHSOM consists of independent GSOMs. For every **node** in a GSOM, a **child GSOM** can be generated. The original GHSOM algorithm uses a *Growing Grid* similar to the authors above [27]. This paper has chosen to use another implantation known as Directed Batch Growing Hierarchical Self-Organizing Map (DBGHSOM) [99]. A major problem with GSOMs is that the map could

grow to be incredibly large, thus causing performance issues and clustering errors. Using vertical, hierarchical growth, the GHSOM can avoid this problem by creating many smaller GSOMs.

**Algorithm 3** DBGHSOM Algorithm

**Input:** Data Dimension (D), Spread Factor (SF), Learning Rate (LR), Total Epochs (T)
**Output:** Weights (W)

```
    BEGIN
    Initialization
1:  Same as Alg. 2
    Horizontal Growing Phase
2:  Same as Alg 2
    Vertical Growing Phase
3:  Calculate the Sum of all CE (SE)
4:  Calculate the Vertical Threshold VT = LR * SE
5:  for All nodes with CE > VT do
6:      Create new child DBGSOM
7:      Train new child DBGSOM using Alg. 2
8:  end for
9:  return W
    END
```

The pseudocode for the hierarchical GSOM used in this paper can be found in Alg. 3. DBGHSOM has 3 phases: initialization, horizontal growing, and vertical growing. The parameters, initialization, and horizontal growing phases are the same as DBGSOM. A 2-by-2 set of nodes is created and initialized. After the first horizontal growing phase, a vertical growth threshold (VT) is calculated. The vertical growth threshold is a percentage of the total cumulative error of a map. For any node $GT_i > VT$, we create a new DBGSOM. This child GSOM is trained just like its parent.

GHSOMs share some of the benefits that its predecessor has, like being able to dynamically grow. Additionally, they have the benefit of both graphically and abstractly represent data in a

hierarchical structure. This form of growth allows for the GHSOM to learn of new attacks as the training data introduces them. The various roots in a GHSOM can be created to have different representations of what constitutes an attack. However, a notable issue with GHSOMs is their ability to grow into thousands of sub-trees and roots, causing performance issues. This problem can be addressed through a pruning process discussed in the next section.

### 2.5 Neural Network Rule Extraction

Rule Extraction (RE) algorithms are a family of techniques used to generate textual rulesets from neural networks. RE can be categorized into three families: decompositional, pedagogical, and eclectic [30]. The decompositional approach opens up the black box and uses neuron weights and activations to generate rules. This approach generates rules layer-to-layer starting from the output and final hidden layer. It then steps backward connecting hidden to its next hidden layer, finally connecting the input layer to the first hidden layer. A substitution algorithm is used to create a ruleset from this chain of rules. Conversely, the pedagogical approach maintains the model's black box nature. It maps model inputs to outputs and uses this mapping to train a DT. Notably, this approach typically trains only a single tree. Lastly, there is the eclectic approach. Eclectic algorithms use a mixture of decompositional and pedagogical techniques. In this type of algorithm, each hidden layer is used to generate its own textual ruleset. These rulesets can be concatenated together to explain the whole of the network. In this work, we chose to use an eclectic-type RE algorithm. The customizable nature of the algorithm allows us to make design choices that benefit IDS datasets.

Eclectic algorithms lead themselves to be more useful for intrusion detection and its large datasets. Decompositional algorithms such as DeepRed [109] have an exponential runtime complexity [107]. This complexity is due to how this algorithm substitutes rules from the input layer to the output layer. However, using an eclectic algorithm, we can mitigate this. We do not necessarily need to use every layer. We can increase the speed of our algorithm by limiting the percentage of layers we generate rulesets for. An important benefit of decomposition approaches is how pure a rule's genealogy is. By this we mean, there is a path through each hidden layer's DT from input to output. This approach may be considered more trustworthy than the eclectic approach. The eclectic approach generates rulesets for each layer. Each layer's weights are correlated with the output of the network. So, unlike the decompositional approach, there is no direct path from input to output.

**Algorithm 4** DNN Rule Extraction

**Input:** Dataset (X), Model (M), Decision Tree Algorithm (DT)
**Output:** Final Ruleset (R)

```
BEGIN
R = set()
Y' = M.predict(X)
for hidden layer h_i in M do
    X' = h_i(X)
    hidden_dt = DT(X', Y')
    Rules_hidden = ExtractRules(hidden_dt)
    Ŷ = Rules_hidden(X')
    input_dt = DT(X, Ŷ)
    R.add(ExtractRules(input_dt))
end for
return R
END
```

The pseudo-code for the algorithm we implemented can be found in Algorithm 4. The algorithm takes input of a dataset (X), a DNN model (M), and a decision tree algorithm (DT). It trains two decision trees per layer that we can extract rules from to generate a ruleset. The algorithm begins by initializing an empty set ($R$) that will be used to store future rules. We then used the trained model to generate predicted labels for our dataset ($Y'$). We then loop through each hidden layer ($h_i$). For each hidden layer, we generate a new dataset ($X'$). This dataset is generated by obtaining the hidden values created by the hidden neurons on each layer. We can then use our hidden value dataset ($X'$) and our predicted labels ($Y'$) to train our hidden value decision tree ($hidden_{dt}$). After the decision tree is trained, we can extract the rules from the tree using a depth-first search approach (ExtractRules()). Once we have our rules extracted, we use the rules to generate a list of labels ($\hat{Y}$). These labels are used in conjunction with the original dataset to create the final *input-to-output* decision tree ($input_{dt}$). The same depth-first search algorithm is used to extract the rules which are then added to the ruleset ($R$). More information about our specific implementation can be found in Section 6.2.3.

# CHAPTER 3

# RESEARCH DATA

This chapter discusses the three intrusion detection datasets used in this dissertation. The three datasets are NSL-KDD, CIC-IDS-2017, and UNSW-NB15. These three datasets are ubiquitously used in the area. The following sections will give brief statitical overviews of each dataset and quickly discuss why they are used in this work. Additionally, each attack is defined so that the reader can form a general understanding of the datasets.

## 3.1 NSL-KDD

NSL-KDD is an intrusion detection dataset created by the University of New Brunswick in 2009 [97]. It is a revised version of the KDD'99 dataset. NSL-KDD seeks to correct issues with the KDD'99 dataset. First, it removes many of the redundant samples. Second, NSL-KDD is made to be more challenging than its predecessor. This dataset was chosen because of its wide use in the literature. Using it allows this work to be compared to past work and can be used to show architecture scalability. This dataset consists of 148,517 samples, 43 features, and a contamination rate of 48.1%. NSL-KDD was synthetically created using a network simulator.

The NSL-KDD dataset contains four different kinds of attacks: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probing. The DoS is an attack that affects the availability of a service, usually causing that service to be unreachable due to too many connection attempts.

However, in general a U2R attack affects the confidentiality or integrity of data. Malicious users, by some means, may gain *root* access to a system and may be able to view or modify data as they please. Similarly, a R2L attack is one where attackers may gain remote access to a machine for which they should not have access. Lastly, probing is a method of information gathering achieved by attackers. This attack looks for known compromised modules that are connected to the internet. While the features (see Table 3.1) of NSL-KDD mainly pertain to tcp/ip packet information, there are also features relating to traffic and content. Traffic features contain information about the duration and amount of connections. Content features contain information about data that the attackers sent in the packet. As mentioned previously, this dataset allows our work to be compared to older IDS and X-IDS works found in the literature. A brief overview of the statistical characteristics of this dataset can be found in Table 5.1.

Table 3.1: Selected features for the NSL-KDD and CIC-IDS-2017 using Bayesian probability of significance [33]

| NSL-KDD | | | |
|---|---|---|---|
| **Feature Name** | **Feature Description** | **Feature Name** | **Feature Description** |
| **src_bytes (Source Bytes)** | Number of bytes from source to destination | **dst_bytes (Destination Bytes)** | Number of bytes from destination to source |
| **Count** | Number of connections to the same host as the current connection at a given interval | **srv_count (Service Count)** | Number of connections to the same service as the current connection at a given interval |
| **dst_host_count (Destination Host Count)** | Number of connections to the same destination | **dst_host_srv_count (Destination Host Service Count)** | Number of connections to the same destination that use the same service |
| **CIC-IDS-2017** | | | |
| **Feature Name** | **Feature Description** | **Feature Name** | **Feature Description** |
| **Flow Bytes/s** | Number of flow bytes per second | **Flow Duration** | Duration of the flow in microsecond |
| **Flow IAT Max (Flow Inter-Arrival Time Max)** | Maximum time between two packets sent in the flow | **Fwd IAT Total (Forward Inter-Arrival Time Total)** | Total time between two packets sent in the forward direction |
| **Flow Packets/s** | Number of flow packets per second | **Destination Port** | Port the package was destined for |
| **Bwd IAT Total (Backward Inter-Arrival Time Total)** | Total time between two packets sent in the backward direction | **Fwd Packets/s (Forward Packets/s)** | Number of forward packets per second |
| **Flow IAT Min (Flow Inter-Arrival Time Min)** | Minimum time between two packets sent in the flow | **Packet Length Variance** | Variance length of a packet |
| **Flow IAT Mean (Flow Inter-Arrival Time Mean)** | Mean time between two packets sent in the flow | **Fwd IAT Max (Forward Inter-Arrival Time Max)** | Maximum time between two packets sent in the forward direction |
| **Idle Max** | Maximum time a flow was idle before becoming active | **Idle Mean** | Mean time a flow was idle before becoming active |
| **Idle Min** | Minimum time a flow was idle before becoming active | **Flow IAT Std Flow (Inter-Arrival Time Standard Deviation)** | Standard deviation time between two packets sent in the flow |
| **Bwd IAT Max (Backward Inter-Arrival Time Max)** | Maximum time between two packets sent in the backward direction | | |

## 3.2 CIC-IDS-2017

CIC-IDS-2017 was created by the same authors of NSL-KDD [90]. The authors evaluated 11 intrusion detection datasets since 1998 and found that these datasets were not reliable. CIC-IDS-2017 was created to feature modern attacks and modern benign network traffic. This dataset is chosen not only because of its wide use in the literature but also because of its size and modern attacks. Compared to the other datasets used for this dissertation, CIC-IDS-2017 is relatively large. It contains 2,827,876 samples, 79 features, and a contamination rate of 19.8%. CIC-IDS-2017 was synthetically created over 5 days.

The dataset includes six types of attacks: Brute Force, Heartbleed, Botnet, Denial of Service, Distributed Denial of Service (DDoS), Web, and Infiltration. A Brute Force attack is a common type of attack whereby a malicious actor tries millions of passwords in an attempt to gain access to a user or administrator account. This type of attack will use resources such as the 'rockyou.txt' common password list and can affect all aspects of the CIA triad. Heartbleed is a confidentiality attack that exploits a weakness in the OpenSSL library [18]. Abusing this bug allows attackers access to encrypted data by reading straight from a compromised system's memory. A Botnet attack refers to the involvement of a set of machines used by an attacker to perform a malicious task. It can affect confidentiality, availability, and integrity. The DDoS attacks differ slightly from the DoS attacks in that this type of attack originates from many different hosts. The Web attacks consist of various SQL injections and Cross-Site Scripting attacks that can affect the confidentiality and integrity of data. Lastly, Infiltration attacks exploit vulnerable software on a user's system to allow an attacker to gain backdoor access potentially affecting all CIA tenets.

### 3.3 UNSW-NB15

The final dataset used in this work is UNSW-NB15. This dataset was created by the University of New South Wales in 2015 [68]. UNSW-NB15 was created to address some of the limitations associated with NSL-KDD. Additionally, it was created with attacks that were modern at the time. This dataset has been chosen because it contains more modern data than NSL-KDD and it has seen some use in the literature. Though the dataset contains over 2 million samples, this work chooses to use UNSW's provided training and testing datasets. Combined, they have a total of 257,637 samples, 45 features, and a contamination rate of 63.9%.

UNSW-NB15 contains 9 types of attacks: fuzzers, analysis, backdoors, DoS, exploits, generic, reconnaissance, shellcode, and worms. Fuzzer attacks use a program that creates random or semi-random inputs that are used to probe network-connected programs for bugs. Analysis attacks employ various port, spam, and HTML attacks. Backdoors are attacks that seek to bypass security layers. Exploits are known security vulnerabilities that can be attacked. Generic attacks are a technique used to attack block-cipher encrypted data. Reconnaissance encompasses all attacks that gather information. Shellcode is a payload attack that attacks known security exploits in programs. Lastly, worms are attacks that replicate themselves in order to spread to other computers.

# CHAPTER 4

# CREATING AN EXPLAINABLE INTRUSION DETECTION SYSTEM USING SELF ORGANIZING MAPS

## 4.1 Introduction

The use of Artificial Intelligence (AI) in cyber-defense solutions, particularly Intrusion Detection Systems (IDS), has been gaining traction to protect against a wide range of cyber attacks. While these AI models have high detection rates, high false positive and false negative rates can dissuade a security analyst from using an AI enabled IDS [63]. These IDS built using AI and deep learning methods are black boxes, meaning a security analyst will have little to no explanations and clarifications on why a model made a particular prediction. With the rise in cyber attacks on critical infrastructure, government organizations, and business networks, there is a pressing need for an explainable, automated detection system that can provide real-time aid to an analyst.

Intrusion Detection Systems are generally utilized as part of a larger cybersecurity defense effort at an organization generally located in a Cyber-Security Operations Center (CSoC). These systems monitor networks and automate attack detection by comparing network activity to the signature of known attacks or by detecting behavior that is anomalous to benign network patterns [83]. Through these methods, a security analyst can use an IDS to detect improper use, unauthorized access, or the abuse of a network. Analysts can then create mitigating strategies to minimize damages and

costs of the malicious behavior. The usefulness and cost effectiveness of IDS have therefore been the subject of much research [13, 103].

Previous work in AI enabled IDS has generally focused on improving detection rates while limiting false positives and false negatives. These techniques have been effective at achieving high detection rate, but have failed to provide explanations for their computed predictions. Without the ability to understand how a model reached a decision and which features were relevant to the decision computation, a security analyst will give less credence to these AI enabled IDS. Opaque Deep Learning methods in particular, can be considered as black boxes providing no explanations and feature relevance information, severely limiting adoption in real world cyber-defense scenarios [57].

A potential solution to this problem is to research and develop Explainable Intrusion Detection Systems (X-IDS) based on current capabilities in Explainable Artificial Intelligence (XAI) [72]. The guidelines proposed by the Defense Advanced Research Projects Agency (DARPA) indicate that to be explainable, an AI should explain the reasoning for its decisions, characterize its strengths and weaknesses, and convey a sense of its future behavior [28]. An X-IDS that is transparent in its behavior and decision making process, will empower a security analyst to make better informed actions, understand the feature composition of a prediction, help CSoCs defend from known attacks, and quickly understand zero-day attacks. To address this need, we create an X-IDS using Self Organizing Maps (SOMs).

Data collected from modern networks contain potentially hundreds of different features about the traffic flow, operating systems, network protocols, and other metadata. SOMs work by representing this high dimensional data on a 2-dimensional plane. This also includes maintaining the

topographical relationship of the data by grouping similar data [47]. Through this dimensional reduction and various other SOM visualization techniques, a security analyst can view both global and local explanations about a potential attack rather than an opaque prediction generated by a black box model.

As the need for explainable cyber-defense systems increases and to address the lack of XAI research in the field of IDS, the main objective of this paper is to demonstrate the explainability of the SOM based X-IDS rather than creating the most accurate system. Higher accuracy systems can be developed by using complex derivative architectures. However, further research is necessary to make them explainable. Our goal in this paper is to increase trust in IDS and help CSoCs defend from attack through the use of explainable insights. As a secondary focus, we also provide the accuracy scores of our SOM based X-IDS system trained on the NSL-KDD and CIC-IDS-2017 datasets.

Major contributions presented in this chapter are -

- A novel X-IDS architecture featuring a SOM, built using DARPA's proposed guidelines for an explainable system. This system is able to produce robust, explanatory visualizations of the SOM model and create accurate IDS predictions.
- A Local and Global explainability analysis using the SOM explainable architecture. The explanation module creates a collection of explainable visualizations that can be used by a security analyst to understand predictions.
- A performative analysis using NSL-KDD and CIC-IDS-2017. The SOM based model is able to achieve accuracies as high as 91% on NSL-KDD and 80% on CIC-IDS-2017 datasets.

## 4.2 X-IDS Architecture

An X-IDS's main goal is to help stakeholders protect their networks and understand various relevant events. The system should act as both a guard and adviser for network security. When an

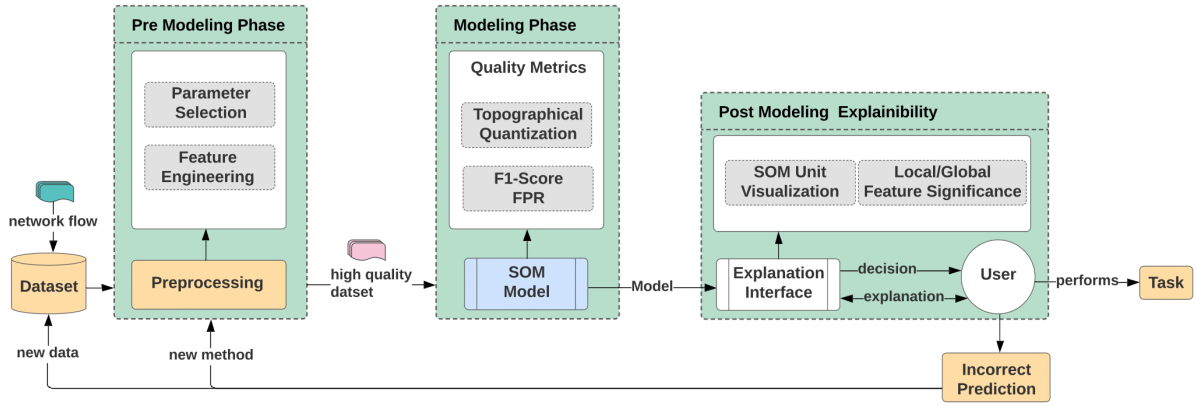



Figure 4.1: Architecture for an Explainable Intrusion Detection System (X-IDS) utilizing Self Organizing Maps (SOMs), based on DARPA's recommended architecture for Explainable Artificial Intelligence (XAI) systems [28].

IDS discovers an intrusion, the user should be notified to prevent a possible attack. Explanations generated by the X-IDS should assist CSoC operators in their mission to protect their organization. To help achieve this goal, we propose the novel, proof of concept SOM based X-IDS architecture in Figure 4.1. The proposed architecture is based on DARPA's recommended architecture for XAI systems [28]. Components of the framework can be changed to suit users' needs. The architecture is abstract enough that methods other than SOMs can be interchanged to create different X-IDS. The architecture consists of three stages: pre-modeling, modeling, and post-modeling explainability. In the first phase, the model preprocesses raw network data captured into high quality datasets, and selects parameters for the SOM model. Next, the model is trained during the modeling phase. Metrics are recorded to determine the newly trained model's quality. Lastly, in the post-modeling phase, the SOM is data-mined to generate explanatory visualizations that allow users to understand how predictions are generated. In the next subsections, we describe each of these phases in detail.

### 4.2.1 Pre-Modeling Phase

In this work, NSL-KDD [98] and CIC-IDS-2017 [80] were used to test the explainability and effectiveness of our architecture. NSL-KDD was chosen because of its wide use in the literature. It allows our method to be compared to other existing IDS. CIC-IDS-2017 includes more modern attacks and is useful for testing an unbalanced dataset. The datasets are passed through a preprocessing module that normalizes the data. Additionally, the architecture uses Bayesian Probability of Significance [33] to select features. Any feature significance value over a designer selected threshold is included in the preprocessed dataset. The resulting high quality dataset is used during the modelling phase.

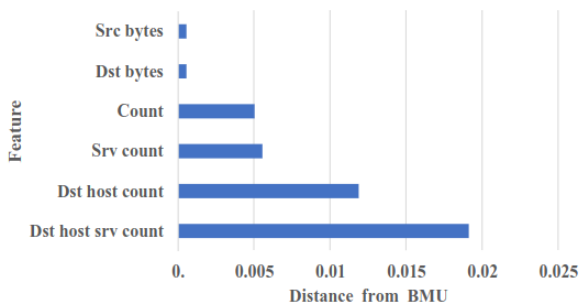
### 4.2.2 Modeling Phase

The modeling phase begins by training the SOM model using the high quality dataset generated during the pre-modeling phase. For this paper, we use the POPSOM implementation [106]. Training parameters include total training iterations, learning rate, and SOM size. At the end of the training session, the model will be tested to produce topographical error, quantization error, F1-score, precision, recall, and a confusion matrix. The confusion matrix can be used to determine both the false positive and false negative rate for the model. The quality metrics are used to determine if a model has been sufficiently trained to generate explanations.

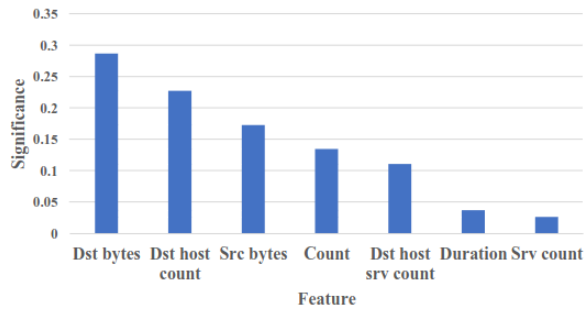
#### 4.2.2.1 Quality Metrics

There have been various metrics and measures proposed to evaluate the quality of a trained SOM. These include quantization error, topographic accuracy, embedding accuracy, and convergence index. Quantization error was used by Kohenen [49], and measures the average distance

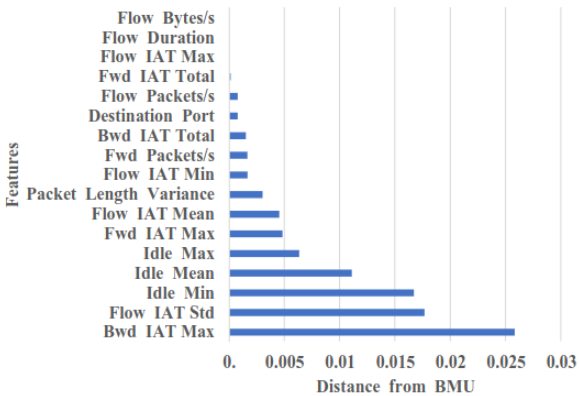
between nodes and the data points. To measure how much the features of the input space have been preserved in low dimensional output space, a topographic error is used. The topographic error is measured by evaluating how often the BMU and second BMU are next to each other [16], [52]. Map embedding accuracy is similar to quantization error and it measures how similar the distribution of the input data is with respect to that of the SOM units [32]. In order to measure both topographic preservation and distribution similarity between the input and SOM units, the convergence index was proposed to be a measure that linearly combines map embedding accuracy and topographic accuracy [1]. Prediction accuracy metrics are also important to include in an IDS architecture. These metrics include F1-score, false positive rate, and false negative rate. These measurements allow the architecture to be compared to other existing IDS.



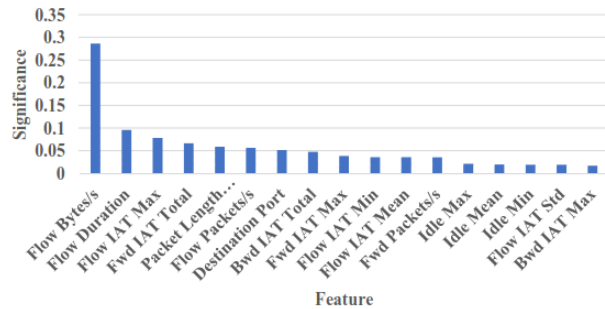

(a) NSL-KDD Local Anomaly Explanation




(b) NSL-KDD Global Feature Significance




(c) CIC-IDS-2017 Local Benign Explanation




(d) CIC-IDS-2017 Global Feature Significance

Figure 4.2: These figures show the local and global feature explanations for both the NSL-KDD and CIC-IDS datasets. (a)(c) Demonstrates features the SOM has chosen for a malicious sample from the NSL-KDD and a benign sample from the CIC-IDS-2017 datasets. The more significant a feature is, the lower its value (i.e. closer to the BMU). (b)(d) Global feature significance is calculated using Bayesian Probability of Significance [33]. Features that have a higher significance value are much more likely to cause a prediction to be made for benign or anomalous.

### 4.2.3 Post-Modeling Explainability

Once the modeling phase has been completed and quality metrics have ensured that the model is a good representation of the data, the model can be used to perform a variety of explainability and visualization tasks. The model itself is a list of SOM units and the weights associated with these units. Visualization tasks include creating local and global explanations, a U-Matrix, and feature heatmaps.

#### 4.2.3.1 Local and Global Explanations

Global and local interpretability can be achieved by examining important features of the trained SOM, and utilizing this information to generate an explanation for a specific data instance classification or cluster classification [101]. Global significance for NSL-KDD is shown in Figure 4.2b with higher values denoting that a feature has a higher probability of being important. Higher variance features increase the probability that a model will capture the dataset's structure [33]. Through this graph, an analyst can understand which features are important to the overall SOM structure, allowing them to examine predictions at a local level based on globally important features.

Figure 4.2a shows the local explanations for a prediction, where each feature on the y-axis has a value representing distance from its respective BMU value. In this example, we can see the features with the largest impact on a prediction: duration, dst bytes, and src bytes. These features were the closest to the BMU, and they played a large role in computing the predicted value. Seeing the specific features that influence predictions provides insight about samples labeled as malicious or benign and can further help operators determine the reason of incorrect predictions. These features can also be further investigated with feature value heat maps.

#### 4.2.3.2 Unified Distance Matrix (U-Matrix)

The U-Matrix is a visualization of the distances between neighboring SOM units. With distances shown as a color gradient, units far apart will create dark boundaries while areas with similar units will be lighter. This can visually represent the natural clusters of input data. To enhance the standard U-Matrix, the starburst model uses connected component lines of nodes overlaid on the matrix to better represent clusters [34]. For a labeled data set, the user is able to visualize each BMU along with the associated label. Figure 4.3a shows clear clusters with boundaries separating malicious (1) and benign (0) behavior. Using this information a security analyst can investigate more visualizations and feature importance values to gain an understanding about why certain malicious network activities are being grouped together.

#### 4.2.3.3 Feature Value Heat Map

Heat maps show general trends that a feature has on the entire SOM model. SOM feature values are represented from 0 to 1, and the heat maps denote this with darker and lighter values, respectively. An example feature value heat map can be found in Figure 4.3b. In this example, the 'dst bytes' features has a cluster of higher values in the bottom-left corner, while the rest of the SOM consists of lower values. Users can use this information to form conclusions about the model. Feature value maps are more powerful when multiple are viewed at a time. In addition, the U-Matrix or K-means clustering charts can then be referenced to make general decisions about the model. The heat maps work well as a fine-grained global explanation that helps users to understand the overall model.

## 4.3 Experimental Design

The SOM based X-IDS was evaluated on both *explanation generation* and *traditional accuracy tests*. SOM explanation generation is categorized into visual and statistical explanations. The visual explanations include U-matrices, feature heatmaps, and K-means clustering maps. Statistical explanations include local and global feature significance charts. These explanations are described in Section 4.2.3. Traditional accuracy tests include many ubiquitous metrics used in AI. This work records F1-score, precision, recall, false positive rate, and false negative rate. All of these metrics are described in Section 4.2.2.

### 4.3.1 Model Parameters & Dataset Preprocessing

Experiments were run using the NSL-KDD and CIC-IDS-2017 datasets (See Chapter 3), which was used to train two 18x18 SOMs. The training process was completed in 1000 iterations over the dataset. After 1000 iterations, there was no significant improvement in evaluation metrics. In fact, while training on the CIC-IDS-2017 dataset, the SOM model performance began to degrade as a result of over-fitting. A learning rate of .3 was chosen as it provided the best clustering results.

The NSL-KDD and CIC-IDS-2017 training and testing datasets are combined to form single datasets. This dataset is then split using Scikit-Learn's **train_test_split()** for a 60% training dataset and a 40% testing dataset. Dataset preprocessing is as follows. First, samples with 'None' or 'NaN' values are removed. Second, categorical entries are One Hot Encoded. Third, the features are normalized using Scikit-Learn's Normalizer on default settings. Lastly, the labels are changed to '0' for benign and '1' for malicious.

### 4.3.2 Explanation Generation

After the models are trained, they can be data-mined to create explanations. Explanation evaluation does not have objective metrics for visual explanations. Users must look at the U-matrices that are generated and form their own conclusions. To this end, this experiment is evaluated using a method that users would use to understand the model. By combining knowledge from both the U-matrices, feature heatmaps, and statistical explanations, this work creates a methodology for potentially evaluating explanations. Local and global statistical explanations, on the other hand, are recorded using defined metrics. Local significance is calculated using the distance from a sample's best matching unit. Global significance is a measure of feature variability in its dataset.

### 4.3.3 Traditional Accuracy Metrics

Traditional accuracy tests cover all of the previously mentioned metrics. F1-score takes into account the model's precision and recall. This help accommodate unbalanced datasets where models may choose to guess the dominant category to achieve high accuracy.

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.1}$$

Precision and recall are metrics that can be used to understand the model's ability to predict true labels. Precision is used to quantify a model's ability to avoid false positives. Recall quantifies

a model's ability to predict true labels as true. In other words, it shows how well the model avoids false negatives. Precision and recall are defined in the equations below.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

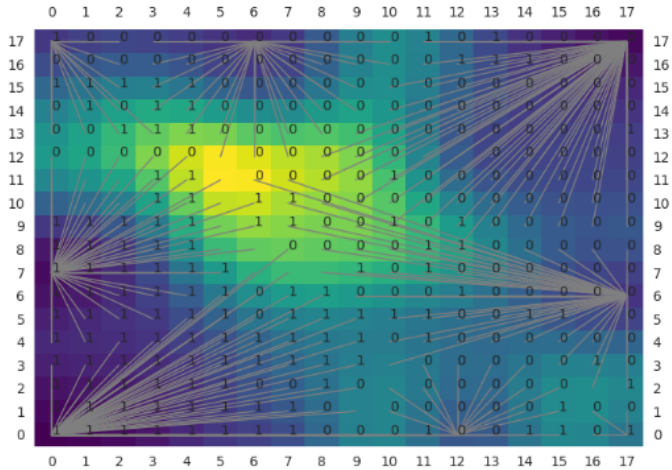
Lastly, False Positive Rate (FPR) and False Negative Rates (FNR) are used to show how often malicious data is labeled as benign and vice versa.

$$FPR = \frac{FP}{FP + TN} \tag{4.4}$$

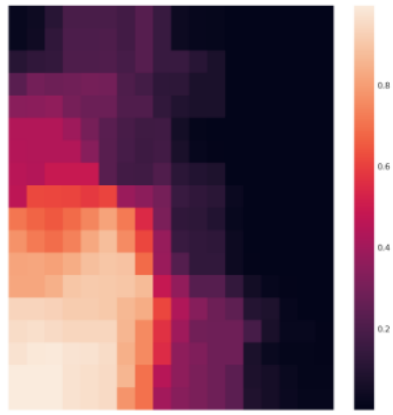
$$FNR = \frac{FN}{TP + FN} \tag{4.5}$$

## 4.4 Experimental Results and Evaluation

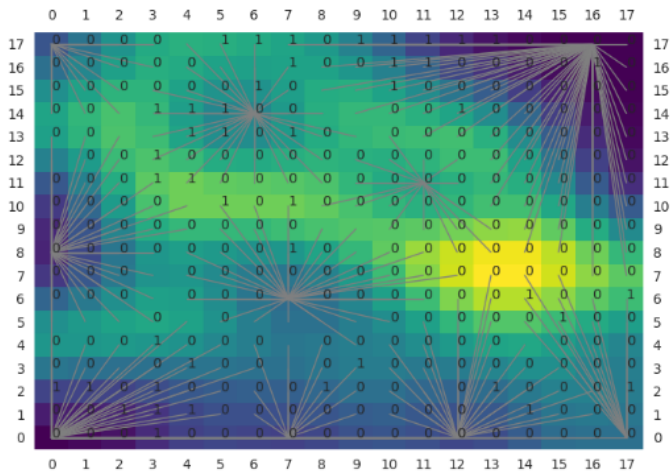
Using the methodology discussed in Section 4.3, this section details the results of the explanation generation and traditional accuracy tests.



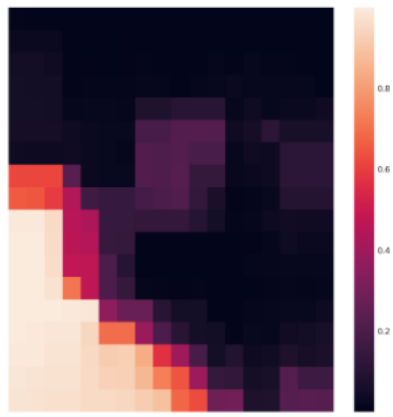
(a) NSL-KDD Starburst U-Matrix



(b) Destination (Dst) byte Feature Map



(c) CIC-IDS-2017 Starburst U-Matrix



(d) Flow bytes/s Feature Map

Figure 4.3: (a)(c)The Starburst U-Matrix shows both the most common label for each node and the clusters the SOM has learned. Darker areas represent units that are close Euclidean Distance-wise. Notably, we can see a clear divide between classes on the NSL-KDD dataset as represented in the figure. (b)(d) The feature value heatmap displays the value of a specific feature on each unit in the SOM. Lighter values represent units with values closer to 1, while darker values show values closer to 0. The ‘dst byte’ example shows that the bottom ‘anomalous’ cluster values higher values.

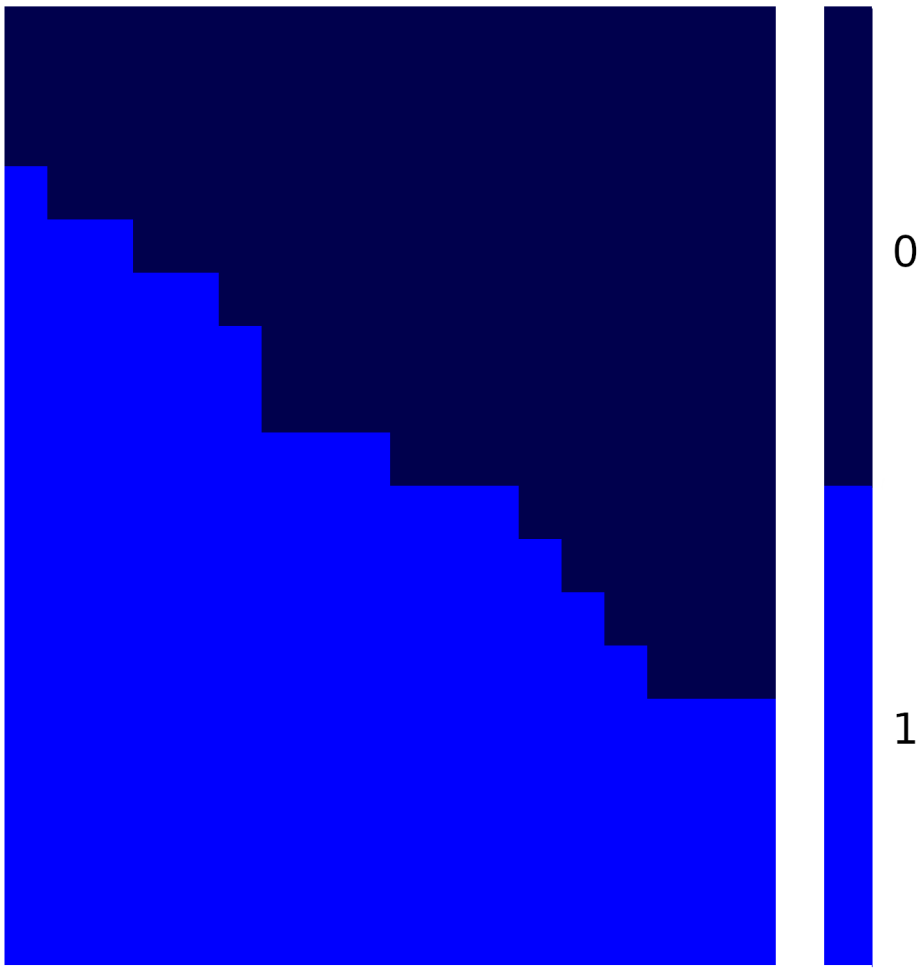



Figure 4.4: NSL-KDD K-means Clustering Map. This visualization can be quickly viewed to summarize where labels cluster on the SOM.

### 4.4.1 Model Explainability

The results for the NSL-KDD dataset can be found in Figures 4.2a and 4.2b. The local explanation example shows that the most important features for its prediction were 'Duration', 'Destination (dst) bytes', and 'Source (src) bytes'. The remaining features, 'Service (srv) count', 'Count', and 'Destination (dst) host count' are considered less significant because of their distance from the BMU. Two of the important features coincide with the Global Feature Significance graph. This trend continues when testing on many different test samples. The most important global features are frequently at the forefront for local significance. Similarly to NSL-KDD, CIC-IDS-2017 follows this trend. Many of the top, globally selected features also play a more important role in the local predictions.

The next set of explainability techniques has been data-mined from the trained SOM. Figures 4.3a and 4.3c show the generated Starburst U-Matrix for NSL-KDD and CIC-IDS-2017, respectively. The SOM algorithm was able to separate benign clusters from malicious clusters in the map created from NSL-KDD dataset. The bottom-left corner is primarily malicious samples, while the top-right corner contains mostly benign samples. Additionally, the clusters marked by the starbursts' origins mostly represent one label. On the other hand, the CIC-IDS-2017 map has not separated the labels sufficiently. Most of the labels present in the figure are benign (0) with very few malicious labels (1). CIC-IDS-2017 is an unbalanced dataset, with about 70% of samples being benign and 30% of samples as malicious. This class imbalance causes the SOM to be trained on more benign samples than malicious.

For a simplified label separation, users can visualize a K-means clustering interpretation in Figure 4.4. This figure helps to explain which clusters the benign and malicious traffic are grouped

Table 4.1: SOM traditional accuracy results table and comparison to black box models.

| Traditional Accuracy Tests | | | | | |
|---|---|---|---|---|---|
| **Dataset** | **F1-Score** | **Precision** | **Recall** | **False Positive Rate** | **False Negative Rate** |
| NSL-KDD | 91.0% | 91.0% | 91.4% | 9.4% | 8.0% |
| CIC-IDS-2017 | 80.0% | 77.4% | 81.8% | 22.5% | 4.5% |

| Peformance Comparison | | | |
|---|---|---|---|
| **Dataset** | **SOM** | **Random Forest** | **Deep Belief Network** |
| NSL-KDD | 91.0% | 99.67% | 97.5% |
| CIC-IDS-2017 | 80.0% | 97.1% | 94.0% |

in. The NSL-KDD K-means graph is similar to the computed U-matrix. However, the CIC-IDS-2017 K-means cluster graph was unable to form accurate clusters. As mentioned above, there were few units that were labeled malicious (1), and the K-means clustering algorithm chosen was unable to create a meaningful separation.

Lastly, the feature value heatmaps are generated for each feature of the dataset. The examples chosen were the most significant features for each dataset: 'destination (dst) bytes' and 'flow bytes/s'. On their own, they can be used to see general training trends for each feature. In Figures 4.3b and 4.3d, we can see that each of these features have higher values in the bottom-left units and lower values elsewhere. Users will be able to build a mental model of the SOM when visualized in conjunction with the features maps. For example, 'destination (dst) byte', 'duration', and 'source (src) byte' all have higher values in the malicious section of the map. One may conclude that when these values are all close to one, the sample is more than likely malicious.

### 4.4.2 Accuracy

When creating an IDS, accuracy is an important metric to consider. Table 4.1 shows the accuracy metrics obtained for both the NSL-KDD and CIC-IDS-2017 datasets. The accuracy of the NSL-KDD evaluation can be attributed to the separation of benign and malicious traffic, as mentioned above. The accuracy of CIC-IDS-2017, however, is much lower. The U-matrix shows that not many units are labeled as malicious. Interestingly, only 14% of the units are labeled as malicious, which means that 77.4% of malicious samples are similar to that small subset of units. Additionally, Table 4.1 compares the SOM's accuracy to Random Forest and Deep Belief Networks (DBN). Here we can see that the SOM performs between 7% to 14% worse than the black box models.

The results from the explainability and accuracy experiments show that it is possible to create explainable and relatively accurate SOM based X-IDS. The visualization techniques used can give users an understanding of the model and can empower security analysts to make their own predictions, similar to the model. A 91% F1-score can be attributed to the clear separation the model makes between malicious and benign samples. When compared to more complex algorithms like Random Forest and DBN, SOM performs worse. However, SOMs are far more explainable and easier to understand. We believe that the accuracy and explainability can be further improved with more complex SOM algorithms.

## 4.5 Conclusion

In this work, we created a novel, proof of concept SOM based X-IDS implementation. The implementation was able to produce robust, explainable figures describing the SOM model. Ex-

plainability was demonstrated using various forms of visualization including feature significance, U-matrices, and feature heatmaps. Through these, users are able to create their own conclusions about how the model works and makes predictions. Additionally, accuracy was tested using the NSL-KDD and CIC-IDS-2017 datasets. The SOM implementation was able to achieve accuracies of 91% and 80%, respectively.

# CHAPTER 5

# EXPLAINABLE INTRUSION DETECTION SYSTEMS USING COMPETITIVE LEARNING TECHNIQUES

## 5.1 Introduction

Shifting away from the current trend of black box Intrusion Detection Systems (IDS) can lead to more trustable and credible anomaly detection. Existing methods for AI enabled intrusion detection use Error Based Learning (EBL) algorithms to detect anomalies. EBL refers to models that train through minimizing a *loss* function, generally through the gradient descent algorithm. These models can achieve high detection rates, however they suffer from a few problems. First, these techniques can impose high performative cost. Neural networks can require both high amounts of time and memory to train [38, 71]. Second, many of these methods have high false positive rates which can harm the overall performance of a real-world IDS [17]. Lastly, these models are not easy to understand and are not innately explainable. Users who use these opaque models do not know how or why a prediction was computed. This can cause a lack of trust and prevent the adoption of AI IDS solutions [4, 63].

eXplainable Intrusion Detection Systems (X-IDS) are a potential solution to the above mentioned problems [71]. To begin, the Defence Advanced Research Projects Agency (DARPA) defines an explainable system as an AI that can explain the *reasoning* for its decisions, characterize its *strengths and weaknesses*, and convey a sense of its *future behavior* [28]. There are many methods

that can allow current EBL AI models to achieve these tenets. Solutions such as Local Interpretable Model-agnostic Explanations (LIME) [85], SHapely Additive exPlantions (SHAP) [60], and Layer-wise Relevance Propagation (LRP) [14] have the ability to convert black box models into semi-transparent, explainable models. However, the use of these types of solutions comes with downsides and a performance overhead. A major downside to these techniques is their black box nature. Similar to the black box EBL models that they are used to explain, the user does not know how or why these explanation frameworks come to conclusions. If one of the goals of black box XAI is to generate trust in opaque models, how can we view explanations from opaque surrogate models as trustworthy? Black Box surrogate explanation can be seen as less trustworthy than certain alternatives. Additionally, creating surrogate models can be a time consuming and resource intensive process. Not only does one need to train the EBL model, but one must also train a secondary surrogate model that generates explanations. White box explanations and white box Competitive Learning (CL) offer a potential solution to these problems. CL algorithms have the benefits of a transparent training process alongside generating explanations during training. Explanations can be data-mined directly from the model and allow for customizable presentation. In other words, CL algorithms are transparent and innately explainable. The transparency of the model and explanations work in tandem to make a more trustworthy system. White box CL algorithms are able to meet all of the criteria DARPA has set for explainable systems, and are a good choice for an X-IDS.

Competitive learning algorithms differ from the more popular EBL algorithms in a number of ways. Where EBL algorithms, such as deep neural networks and recurrent neural networks, learn by adjusting weights to minimize loss, CL algorithms learn through a competitive process. These

CL based techniques, for instance, pits nodes that mimic samples of data against one another. When a node wins in this competition, its weights are adjusted to be similar to the training sample. This enables CL-based techniques to learn by creating abstract representations of data. For IDS datasets, an X-IDS system built using CL models can learn different kinds of attacks and benign behaviors. An important benefit to this type of learning is its ability to be data-mined for visual and statistical explanations. This is a benefit that EBL algorithms lack. See Section 5.2.4 for more information about CL explanations.

The most ubiquitous CL algorithm one will find is the Self Organizing Map (SOM) [47] and its variants. These algorithms consist of a grid of orthogonally connected nodes that each contain a representation of data. As mentioned previously, these nodes compete against one another to mimic abstract patterns in data. The original SOM algorithm consists of a static map of nodes that confine the training area. The Growing Self Organizing Map (GSOM) [27] and Growing Hierarchical Self Organizing Map (GHSOM) [26] improve upon the original SOM design by growing the map horizontally and vertically. These expansion strategies allow the improved algorithms to learn various abstract representations of data than the original SOM. These algorithms are detailed further in Chapter 2.

There are many benefits to using CL algorithms for an X-IDS. As mentioned previously, the CL algorithm's transparency and innate explainability make them much more trustworthy than black box alternatives. Users of an IDS can use CL models to formulate better responses for *tasks* they must perform. These tasks can be performed with confidence because of the trust generated from the CL algorithm and explanations. Security analysts can use the model's explanations to better understand attacks in order to better protect their network. Machine learning engineers may also

be able to discover deficiencies in the model's logic. Using this knowledge, they can modify the architecture or introduce new training samples to increase its overall effectiveness. Explanations will also lead to increasing the IDS's trust and credibility. This can make users more confident that they will be able to complete their tasks.

In this work, we detail our customizable X-IDS architecture that leverages CL algorithms to create explanations and accurate predictions. The architecture consists of four phases: Pre-modeling, Modeling, Post-Modeling Optimizations, and Prediction Explanation. Users are encouraged to modify the architecture when they receive explanations that are not helpful. We then compare the accuracy of the CL models used in our X-IDS architecture to other EBL models and discuss explanations generated by the SOM, GSOM, and GHSOM. We find that our models can have comparable accuracy to black box EBL models, and also have the major benefit of being innately explainable.

Major contributions presented in this work are -

- An X-IDS architecture featuring three CL-based algorithms, built using DARPA's guidelines for an explainable system. Self-Organizing Map, Growing Self Organizing Map, and Growing Hierarchical Self Organizing Map models are used to create explanatory visualizations and accurate predictions. We find that the innately explainable CL models have comparable accuracy to EBL models and that CL explanations can be more trustworthy than their black box counterparts.
- An analysis of statistical and visual explanations for an effective X-IDS. Our X-IDS architecture generates a collection of explainable visualizations ranging from global significance charts to fine-grained feature explanations. Users can use these explanations to understand how and why the model makes decisions.
- A pruning process that can significantly reduce the size of the GHSOM model. The GHSOM algorithm can create more maps than a human can have time to understand. Additionally, larger maps can impose a higher performative cost. The pruning process is able to remove less important branches to improve overall prediction speed while losing little accuracy.
- A performative analysis of our architecture using traditional accuracy metrics. We compare CL models to existing EBL models using the NSL-KDD and CIC-IDS-2017 datasets. CL

models are 1% - 3% less accurate than EBL algorithms. Even though they are less accurate, their innate explainability and trustability make CL algorithms an important tool for X-IDSs.

## 5.2 Competitive Learning X-IDS Architecture

Explanations generated by the X-IDS should assist Cyber Security Operation Center (CSoC) operators in their mission to protect their organization. To help achieve this goal, we create the proof of concept Competitive Learning (CL) based X-IDS architecture in Figure 5.1. The proposed architecture is based on DARPA's recommended architecture for XAI systems [28]. Components of the framework can be changed to suit user's needs. The architecture is abstract enough, such that methods other than CL algorithms can be interchanged to create different X-IDS. The architecture consists of four phases: pre-modeling, modeling, post-modeling optimization, and prediction explanation. In the pre-modeling phase, raw datasets are preprocessed and parameters are selected for the model. In the modeling phase, our CL algorithms are trained and quality metrics are recorded. In our proof of concept system, we are using the SOM family of CL algorithms. In the post-modeling optimization phase, models can then be optimized through various means described below. In the prediction explanation phase, data mining techniques are employed on the resulting models to generate explanatory visualizations that allow users to understand how predictions are generated.

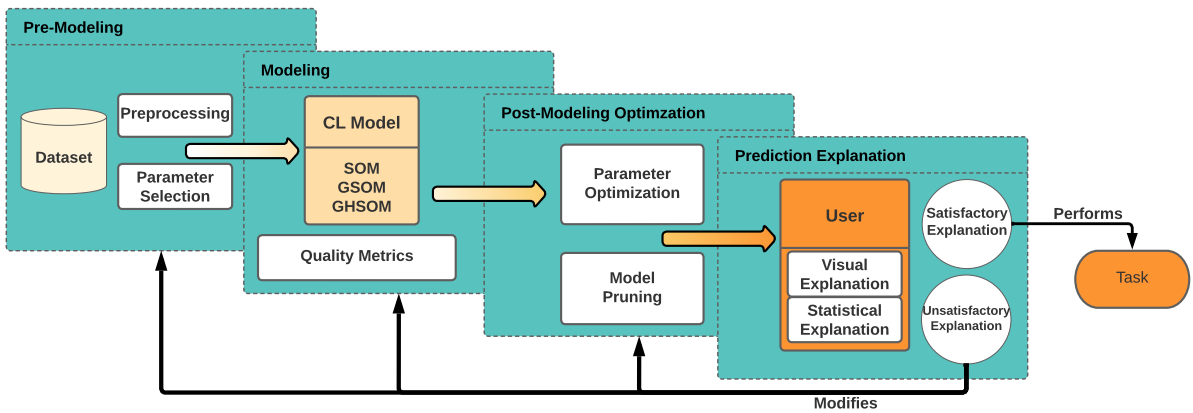



Figure 5.1: A competitive learning based X-IDS architecture. The architecture is divided into four phases: Pre-Modeling, Modeling, Post-Modeling Optimization, and Prediction Explanation. Each phase contributes to translating raw input data into accurate predictions and useful explanations. Culminating in a user successfully completing an associated task or being required to make changes to previous steps in the architecture.

Table 5.1: Statistical characteristics for the NSL-KDD and CIC-IDS-2017 datasets.

| | **NSL-KDD** | **CIC-IDS-2017** |
|---|---|---|
| **Samples** | 148,518 | 2,830,743 |
| **Contamination Rate** | 46.5% | 19.7% |
| **Features** | 41 | 79 |
| **Unique Attacks** | 4 | 6 |
| **Year Created** | 1999 | 2017 |

### 5.2.1 Pre-Modeling Phase

The pre-modeling phase consists of preprocessing raw datasets and initial parameter selection. The preprocessing for our models includes feature selection and normalization. The feature selection algorithm that we have chosen to use is the 'Bayesian probability of significance' [33], which selects the most relevant features from each dataset. Feature selection is not used when training the GHSOM. The GHSOM is able to use all of the features in a datset more effectively due to its hierarchical nature. Additionally, the datasets are preprocessed for binary classification. Lastly, the datasets are normalized to minimize feature bias and improve accuracy. After preprocessing is finished, the new, high-quality dataset can then be passed to the model. The next section details information about the our selected datasets and their usefulness in testing IDS.

#### 5.2.1.1 Model Parameters

The SOM parameters consist of $n$ x $m$ rows and columns, the number of training epochs, and a learning rate (LR). Picking the size of $n$ and $m$ is one of the most important decisions. If the values are too small, clusters will not separate and can begin to merge. If the values are too large, proper clusters may not form at all and resources can be wasted. Determining what this parameter should

be is done through trial and error. Similarly, the number of training epochs can cause significant performance increases. Both scenarios, the one with too many epochs and the one with too few epochs, can cause overfitting and underfitting, or simply waste time. Lastly, LR values affect the learning rate function. A higher value will cause larger changes when adjusting Best Matching Units (BMUs) towards training samples. If the learning rate is set too high, BMU values will mimic singular instances of data. This leads to a less abstract understanding of the data that can cause decreases in accuracy. Too low a value will cause the model to not learn at all. In our testing, we found that a moderately low LR worked best for our datasets.

GSOM and GHSOM contain different parameters than the SOM. These algorithms no longer need to have the number of rows and columns defined. They always start with a 2 x 2 set of nodes. A new, important parameter is the Spread Factor (SF). The SF determines how much Cumulative Error (CE) (see Chapter 2) is needed to create a new, horizontal or vertical node. Smaller SF values cause more node generation. LR is used for the same purpose as in the SOM. It determines how much a BMU will change with regard to the training samples. However, the GHSOM uses LR to also dictate the vertical growth threshold. The pruned GHSOM includes an additional parameter $\delta$ which determines how aggressively the model is pruned. Table 5.2 contains all of our experimental settings.

Data preprocessing and parameter selection are difficult processes that require many iterations to attain the best results. In our work, we have determined that minimal preprocessing is needed to attain high performative results. Initial parameter selection can be done by hand, while later, the user relies on search algorithms depending on the architecture. Both GSOM and GHSOM

Table 5.2: Model Training Parameters for NSL-KDD and CIC-IDS-2017 Models.

| | **Parameter** | **NSL-KDD** | **CIC-IDS-2017** |
|---|---|---|---|
| **SOM** | n | 18 | 18 |
| | m | 18 | 18 |
| | LR | .3 | .3 |
| | Epochs | 1000 | 1000 |
| **GSOM** | LR | .006 | .006 |
| | SF | .9 | .9 |
| | Epochs | 100 | 40 |
| **GHSOM** | LR | .006 | .006 |
| | SF | .3 | .3 |
| | Epochs | 100 | 40 |
| | $\delta$ | .3 | .3 |

have been optimized using a process outlined in Section 5.2.3. The next phase will take the newly preprocessed dataset and selected parameters to create well trained models.

### 5.2.2 Modeling Phase

Using the high quality dataset and the parameters selected in the pre-modeling phase, we can train the set of CL models. We utilize a subclass of CL models described in Chapter 2 which includes three variants of the self organizing map algorithm: the Self Organizing Map (SOM), the Growing Self Organizing Map (GSOM), and the Growing Hierarchical Self Organizing Map (GHSOM). These algorithms create clusters mimicking input data, and in doing so, they create a map that can be data-mined for explanatory purposes. The SOM is the most basic of the three algorithms. It consists of a grid of nodes with no logic to grow and change shape. While, this allows the algorithm to be more easily understood, it can lead to poorer performative results. The GSOM and GHSOM algorithms address this issue by allowing the map of nodes to grow horizontally

and vertically. These algorithms are able to understand more complex structures in data, and their growing nature helps to accommodate these new structures. Please see Chapter 2 for a more in depth description of these algorithms.

#### 5.2.2.1 Model Evaluation Metrics & Techniques

There have been various metrics and measures proposed to evaluate the quality of a trained SOM. These include quantization error, topographic error, embedding accuracy, and convergence index. The quantization error was used by Kohenen [49], and measures the average distance between nodes and the data points. The topographic error measures how well preserved features are in the low dimensional output space. It is measured by evaluating how often the BMU and the second BMU are next to each other [16, 52]. The map embedding accuracy is similar to the quantization error. This metric measures how similar the distribution of the input data is with respect to that of the SOM units [32]. In order to measure both topographic preservation and distribution similarity between the input and SOM units, the convergence index was proposed to be a measure that linearly combines the map embedding accuracy and the topographic error [1]. Performative metrics are also important to include in an IDS architecture. These metrics include accuracy, F1-score, false positive rate, and false negative rate. We also opt to include training time and prediction speed since they can play an important role in intrusion detection. The experimental results using these performative metrics can be view in Section 5.4. These measurements allow the architecture to be compared to the architecture of other existing IDS.

### 5.2.3 Post-Modeling Optimization Phase

Selecting the best parameters for a ML model is an important challenge. However, doing such work by hand is a time consuming process. This is especially true when training times begin to scale higher. Luckily, there are methods that can automate this process. Additionally, some models may also benefit from post-processing. Many tree based models will use a form of post-processing known as pruning to reduce the chance of overfitting and speed up decisions. In this work, we have chosen two techniques for post-modeling optimization. A Bayesian search process is employed to find the parameters for improved performative results, and a pruning technique is used to optimize the size of the GHSOM.

#### 5.2.3.1 Parameter Optimization

There are a few notable methods that can be used for parameter optimization. For this work, we considered Grid search, Random search, and Bayesian search. The differences in these search algorithms relate to how each selects its next set of test parameters. Grid search acts like a brute force approach. It tests as many combinations of parameters as possible. This process is time consuming, but one is more likely to find the best parameters for their model. Random search differs from this approach by sampling a specified number of time from a range of values. This approach limits the number of options that are tested but does not check the entire space of parameter combinations. The last option and the one chosen for our architecture is the Bayesian search. This method limits the search space by creating a surrogate probability model. It makes informed decisions about each set of parameters tested. On average, Bayesian search is able to find

a better set of parameters faster than the other two algorithms. The trade-off is that it may not find the best set of parameters as it doesn't search the entire parameter space.

#### 5.2.3.2 GHSOM Model Pruning

Model pruning can be a valuable resource for certain algorithms. GHSOMs can benefit from pruning by reducing the number of nodes needed to match against. Another notable reason to prune a GHSOM is for easier visualization. GHSOMs can grow into hundreds or thousands of branches making visualization a daunting task. The goal of pruning a tree is to limit its size while also retaining as much information as possible. As mentioned in Chapter 2, we use a pessimistic pruning approach. It is a bottom-up approach that determines whether each node should be kept. To prune a node, an error rate comparison is made. When the error rate for a node is high in comparison to the average error rate, it is removed from the tree. Since this is a bottom-up approach, a node and all of its children can be removed. The visualization changes can be seen in Figures 5.3 and 5.4.

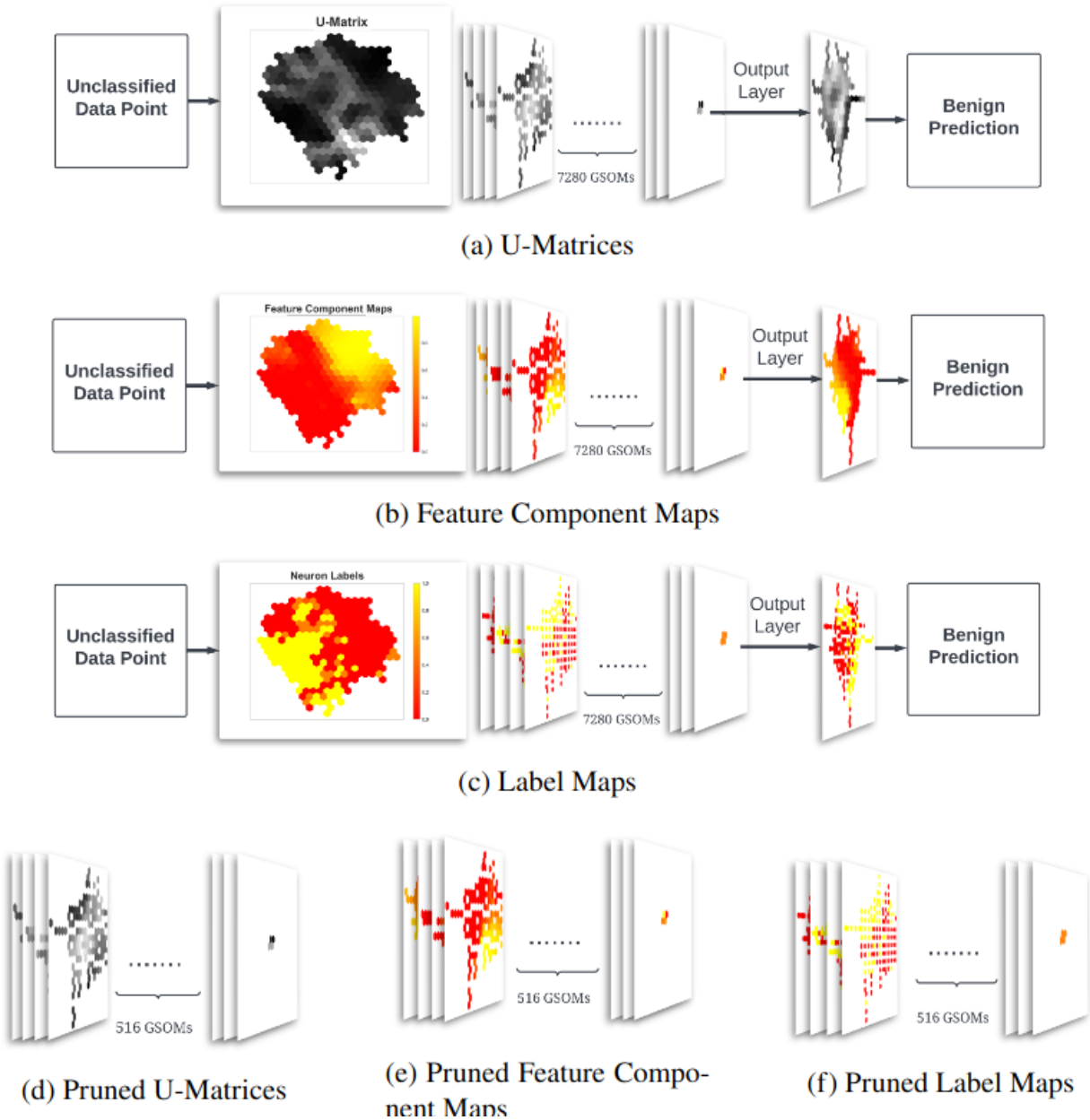



Figure 5.2: Visualizations created from GHSOM trained on NSL-KDD. The left hand unslanted visualizations represent the root GSOM. The slanted visualizations represent GSOMs deeper in the hierarchy of GHSOM. Figure (a) shows the Unified Distance Matrices (U-matrices), which shows the distance between nodes with darker areas representing nodes closer together and lighter nodes representing further distances. Figure (b) shows the feature component maps representing the values of specific features on each node in the GSOM. Figure (c) is the Label maps which show the class labels of the node. Figures (d), (e), and (f) represent the pruned versions of the GHSOMs with significantly less network sizes.

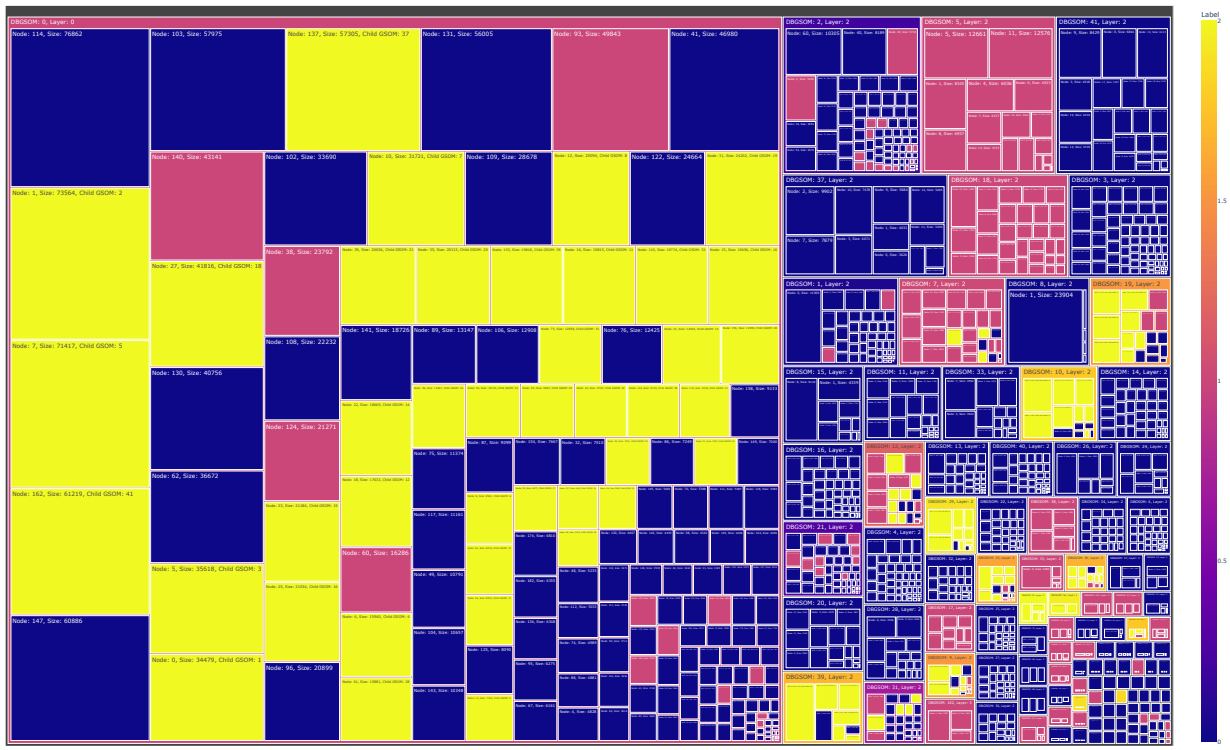



Figure 5.3: This figure contains the results from the trained GHSOM on the CIC-IDS-2017 dataset. Since a GHSOM consists of many GSOMs, The tree map diagram displays GSOMs and their nodes. In this tree map, the left half of the map is the root GSOM. Within the root GSOM we can see a mixture of blue, red and yellow nodes. Blue nodes indicate a benign label, red nodes indicate a malicious label and yellow indicate a branch. The size of each node indicates the number of times it was chosen as the BMU.

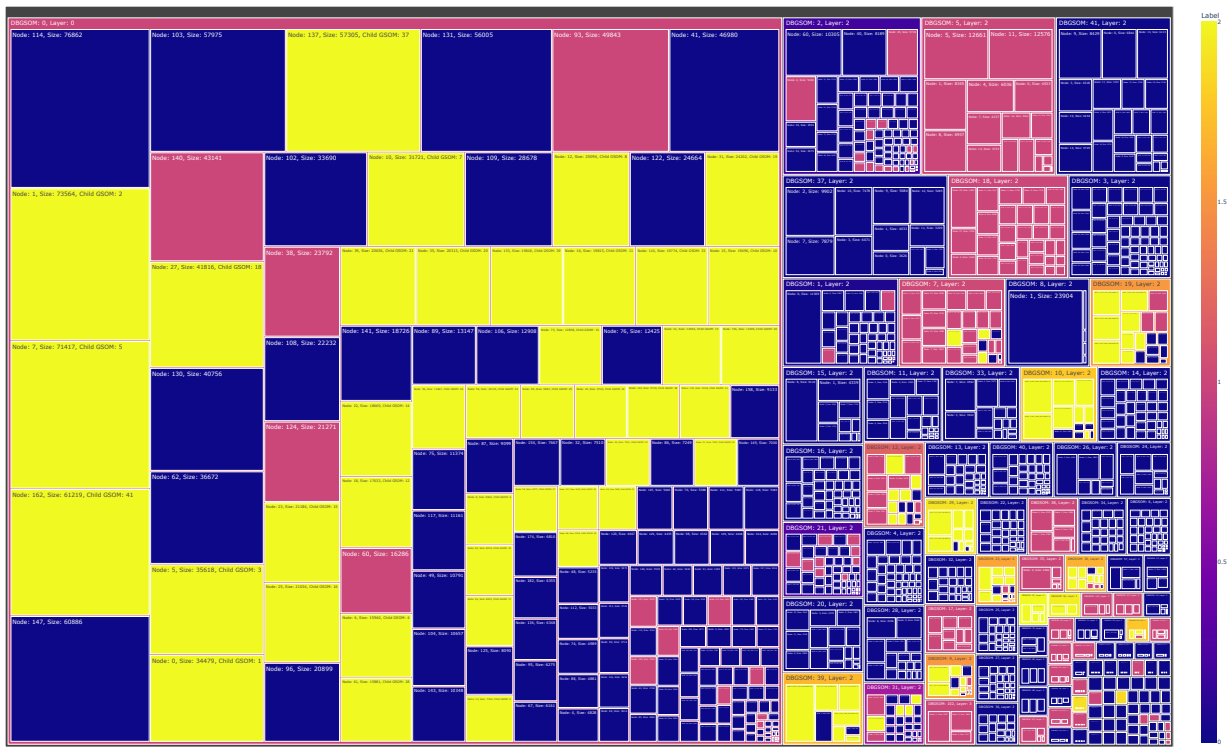



Figure 5.4: The tree map generated after pruning the GHSOM from Figure 5.3. In the previous tree map, nodes would eventually become too small to see. The pruning process outlined in Section 5.2.3 allows the user to see the majority of the nodes.

Optimized models are important for both detection rates and performance. The benefits from this phase can be seen in Table 5.3. Using parameter optimization, we are able to increase the accuracy of our models, while the pruning process allows for faster predictions. These are both important factors when a system is being used for network security.

### 5.2.4 Prediction Explanation Phase

Once the modeling and optimization phases have been completed, and the quality metrics have ensured that the model is a good representation of the data, the model can be used to perform a variety of explainability and visualizations. The models themselves are lists of SOM nodes and the weights associated with these nodes. Visualizations include creating local and global explanations, U-Matrices, and feature heatmaps. Users can use explanations to perform tasks to better defend the network. When a user receives a subpar explanation, the user can modify the architecture where needed to help bolster the X-IDS. By using the explanations generated from the white box CL models, the user to build trust and confidence that the model is working as intended.

#### 5.2.4.1 Local and Global Explanations

Global and local explainability can be achieved by examining important features of the trained SOM, and utilizing this information to generate an explanation for a specific data instance classification or cluster classification [101]. Global significance for NSL-KDD is shown in Figure 5.5b with higher values denoting that a feature has a higher probability of being important. The algorithm chosen to determine this variance was 'Bayesian probability of significance' [33]. Higher variance features increase the probability that a model will capture the dataset's structure. Through

this graph, an analyst can understand which features are important to the overall SOM structure, allowing them to examine predictions at a local level based on globally important features.

Figure 5.5a shows the GSOM local explanations for a prediction on the NSL-KDD dataset. Each feature has a value representing the significance. Significance (S) is a calculation involving the *min-maxed* distances from a BMU inverted so that higher values are more important. The formula can be seen in Formula 5.1. In this example, we can see the features with the largest impact on a prediction: destination (dst) host count, duration, and destination (dst) bytes. These features were the closest to the BMU, therefore, they played a large role in computing the predicted value. Seeing the specific features that influence predictions provides insight into samples labeled as malicious or benign and can further help users determine the reason for incorrect predictions. These features can also be further investigated with feature value heat maps.

$$S = 1 - (\frac{X - X_{min}}{X_{max} - X_{min}}) \tag{5.1}$$

#### 5.2.4.2 Unified Distance Matrix (U-Matrix)

The U-Matrix visualizes the distances between neighboring SOM nodes. With distances shown as a color gradient, nodes far apart will create light boundaries while areas with similar nodes will be darker. This can visually represent the natural clusters of input data. To enhance the standard U-Matrix, the starburst model uses connected component lines of nodes overlaid on the matrix to better represent clusters [34]. For a labeled data set, the user is able to visualize each BMU along with the associated label. Figure 5.6a shows clear clusters with boundaries separating malicious (1) and benign (0) behavior. Using this information a users can investigate more visualizations and

feature importance values to gain an understanding of why certain malicious network activities are being grouped together.

#### 5.2.4.3 Feature Value Heat Map

A heat map applied to a feature shows general trends that a feature has on a model, in this case the entire SOM model. SOM feature values are represented from 0 to 1, and the heat maps denote this with darker and lighter values, respectively. An example feature value heat map can be found in Figure 5.6b. In this example, the 'dst bytes' feature has a cluster of higher values in the top-right corner, while the rest of the SOM consists of lower values. Users can use this information to form conclusions about the model. Feature value maps are more powerful when multiple are viewed at a time. The U-Matrix chart can then be referenced to make general decisions about the model. The heat maps work well as a fine-grained global explanation that helps users to understand the overall model.

#### 5.2.4.4 Users Performing Tasks

An important component of our architecture is its *user-in-the-loop* system. A 'user' is a network's stakeholder. There can be many kinds of stakeholders for an IDS. AI engineers who implement and maintain the X-IDS architecture, security analysts who protect the network, and investors manage security expenses. Tasks are performed with the goal of protecting the network and are enhanced by the X-IDS's generated predictions.

When a satisfactory prediction has been created, a user can perform their task. Satisfactory explanations will cause the user to be able to perform their tasks more effectively. However, not all explanations will be useful. When an unsatisfactory explanation is created, a user can use

that explanation to make changes to the parts of the architecture. This could be accomplished by changing how datasets are preprocessed, choosing a new ML model, modifying optimizations, or creating a new style of explanation.

Prediction explanation plays a pivotal role in an X-IDS architecture. Visualizations and statistics are critical for creating actionable explanations for a network. Local explanations can help programmers and security analysts to fine-tune the model, while global explanations can be used by investors to understand the model at a high level. Users can use predictions to either defend the network, or fortify the intrusion detection system. In the next section, we demonstrate the performative results of our models using the above described architecture.

## 5.3 Experimental Design

The SOM based X-IDS was evaluated on both *explanation generation* and *traditional accuracy tests*. SOM explanation generation is categorized into visual and statistical explanations. The visual explanations include U-matrices, feature heatmaps, and K-means clustering maps. Statistical explanations include local and global feature significance charts. These explanations are described in Section 4.2.3. Traditional accuracy tests include many ubiquitous metrics used in AI. This work records F1-score, precision, recall, false positive rate, and false negative rate. All of these metrics are described in Section 4.2.2.

### 5.3.1 Model Parameters & Dataset Preprocessing

The parameters selected for our models can be found in Table 5.2. In this work, the SOM was setup to run over 1000 epochs using an $18\,x\,18$ map. We found that increasing the number of epochs served to overfit the models and decrease the overall efficacy of the model. Both the size of the map

and the number of training epochs were chosen through trial and error. The GSOM parameters were set to 100 and 40 training epochs for NSL-KDD and CIC-IDS-2017 respectively. We found that this in addition to an aggressive Spread Factor (SF) of .9 created the best performative results. The GHSOM parameters were set to 100 epochs per GSOM created using a SF of .3 and a Learning Rate (LR) of .006. These settings were discovered using the parameter selection process outlined in Section 5.2.4. Using these parameters, we were able to create well trained, highly accurate models.

The NSL-KDD and CIC-IDS-2017 training and testing datasets are combined to form single datasets. This dataset is then split using Scikit-Learn's **train_test_split()** for a 60% training dataset and a 40% testing dataset. Dataset preprocessing is as follows. First, samples with 'None' or 'NaN' values are removed. Second, categorical entries are One Hot Encoded. Third, the features are normalized using Scikit-Learn's Normalizer on default settings. Lastly, the labels are changed to '0' for benign and '1' for malicious.

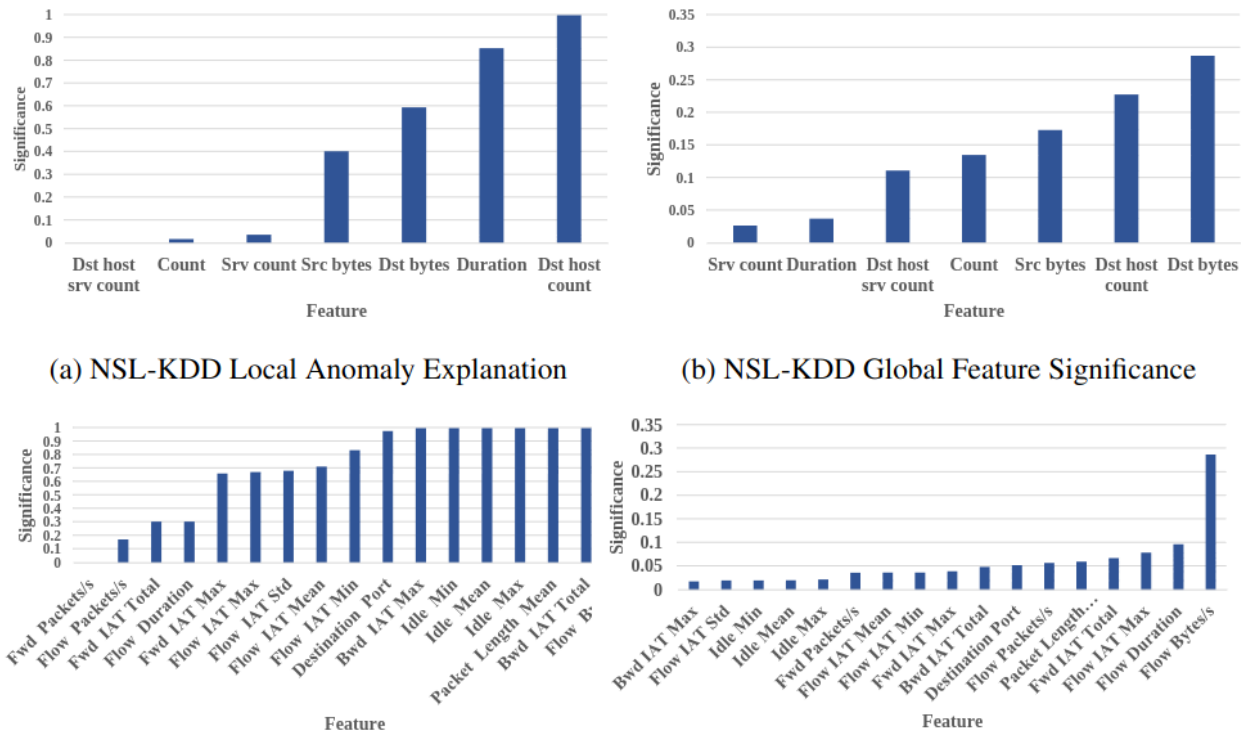
### 5.3.2 Explanation Generation

After the models are trained, they can be data-mined to create explanations. Explanation evaluation does not have objective metrics for visual explanations. Users must look at the U-matrices that are generated and form their own conclusions. To this end, this experiment is evaluated using a method that users would use to understand the model. By combining knowledge from both the U-matrices, feature heatmaps, and statistical explanations, this work creates a methodology for potentially evaluating explanations. Local and global statistical explanations, on the other hand, are recorded using defined metrics. Local significance is calculated using the

distance from a sample's best matching unit. Global significance is a measure of feature variability in its dataset.

### 5.3.3 Traditional Performative Tests

The experimental results consist of accuracy, precision, recall, f1, false positive rate, false negative rate, and network size measures. Accuracy refers to the percentage of correct predictions compared to the total test size. Precision measures the ratio of true positive predictions to the total number of positive predictions. Recall is the measure of true positive predictions to the total number of positive samples in the test set. The f1 score is a measure that gives equal weight of precision and recall. False positive rate is the rate of false positive predictions compared to the amount of ground truth negatives. False negative rate is the rate of false negative predictions compared to the amount of ground truth positives. Network size is simply the amount of GSOMs within the hierarchical GHSOM structure. For SOMs and GSOMs, the network size is 1. The training time of each algorithm and the average time for a single prediction is also measured. All results can be found in Table 5.3.




(a) NSL-KDD Local Anomaly Explanation

(b) NSL-KDD Global Feature Significance

(c) CIC-IDS-2017 Local Anomalous Explanation

(d) CIC-IDS-2017 Global Feature Significance

Figure 5.5: These figures show the local and global feature explanations for both the NSL-KDD and CIC-IDS datasets. (a)(c) Demonstrates features the GSOM has chosen for a malicious sample from the NSL-KDD and CIC-IDS-2017 datasets. The more significant a feature is, the higher its value. (b)(d) Global feature significance is calculated using Bayesian Probability of Significance [33]. Features that have a higher significance value are much more likely to cause a prediction to be made for benign or anomalous.

Table 5.3: Competitive Learning X-IDS results compared to state-of-the-art black box models.

| NSL-KDD | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **SOM** | **GSOM** | **GHSOM** | **P-GHSOM** | **NDNN Jia et al. [41]** | **CNN Mohammadpour et al. [66]** | **BGRU+MLP Xu et al. [104]** | **BAT-MC Su et al. [93]** |
| **Accuracy** | 90.9% | 96.7% | 98.2% | 98.0% | 95.0% | 99.8% | 99.3% | 99.2% |
| **Precision** | 97.2% | 96.6% | 98.0% | 98.0% | - | - | - | - |
| **Recall** | 83.3% | 96.5% | 98.3% | 97.8% | 97.4% | - | 99.3% | - |
| **F1** | 89.7% | 96.6% | 98.1% | 97.9% | 91.4% | - | - | - |
| **FPR** | 2.2% | 3.1% | 1.9% | 1.8% | - | - | 0.8% | - |
| **FNR** | 16.6% | 3.5% | 1.6 | 2.2% | - | - | - | - |
| **Network Size** | 1 | 1 | 7288 | 574 | - | - | - | - |
| **Training Time (s)** | 8 | 60 | 692 | 816 | - | - | - | - |
| **Prediction Time (ms)** | .03 | .03 | .06 | .04 | - | - | - | - |

| CIC-IDS-2017 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **SOM** | **GSOM** | **GHSOM** | **P-GHSOM** | **SDCNN Khan et al. [44]** | **DNN+RE Almutlaq et al. [7]** | **SS-Deep-ID Abdel-Basset et al. [3]** | **CNN-IDS* Halbouni et al. [31]** |
| **Accuracy** | 79.4% | 94.6% | 96.7% | 95.7% | 99.3% | 97.4% | 99.6% | 99.6% |
| **Precision** | 83.2% | 83.7% | 89.1% | 86.5% | 99.1% | 98.3% | 99.5% | 99.7% |
| **Recall** | 42.0% | 90.0% | 94.5% | 92.7% | 99.7% | 99.2% | 99.2% | 99.4% |
| **F1** | 55.8% | 86.7% | 91.7% | 89.5% | 99.4% | 98.3% | 99.4% | 99.7% |
| **FPR** | 19.0% | 4.3% | 2.8% | 3.5% | 1.0% | - | 0.7% | 0.5% |
| **FNR** | 23.0% | 10.0% | 5.5% | 7.3% | 1.0% | - | 0.5% | - |
| **Network Size** | 1 | 1 | 16894 | 119 | - | - | - | - |
| **Training Time (s)** | 260 | 1820 | 4299 | 11205 | - | - | - | - |
| **Prediction Time (ms)** | .03 | .06 | 1.5 | .03 | - | - | - | - |

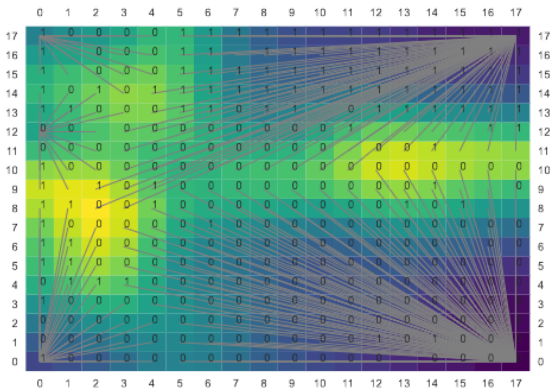
## 5.4 Experimental Results & Evaluation

Our CL based architecture and its SOM variants were evaluated on both *traditional performative tests* and *explanation generation*. The datasets used to test our architecture were NSL-KDD and CIC-IDS-2017 (see Chapter 3). In this section, we examine the performative results from all our CL models, and the explanation results from the Growing Self Organizing Map (GSOM) model. The GSOM explanations were chosen because of the high accuracy of the GSOM model and its similarity to both the SOM and GHSOM.

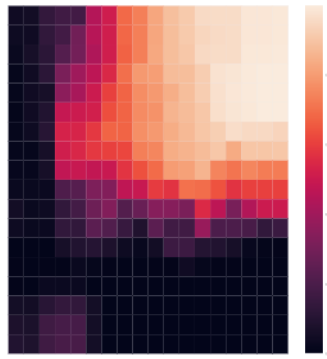
### 5.4.1 Performative Results

The first part of our experiments looked at how accurate our architecture could be. The SOM performed the worst out of our set of CL algorithms. This is expected as it is the least complex of the three algorithms. It achieved an accuracy of 90.9% on NSL-KDD and 79.4% on CIC-IDS-2017. The majority of its accuracy loss comes from its high false negative rate for NSL-KDD and both error rates for CIC-IDS-2017. Out of all of the models, the SOM is the fastest to train and predict with. This may not be a good trade-off, however, with how low its accuracy is. On the other hand, the GSOM, using a more complex algorithm, is able to achieve a much higher accuracy of 96.7% on the NSL-KDD dataset. Additionally, it also greatly improved in accuracy of the CIC-IDS-2017 dataset. The GSOM's growing nature allows it to adapt to new benign and malicious behavior, reducing the number of false negatives. The gain in accuracy comes with the cost of additional training time. Finally, we tested the GHSOM with and without feature selection. The feature selected GHSOM produces an accuracy of 96.2% and 96.1% for the normal and pruned variants on the NSL-KDD dataset. The CIC-IDS-2017 feature selected models both achieved accuracies

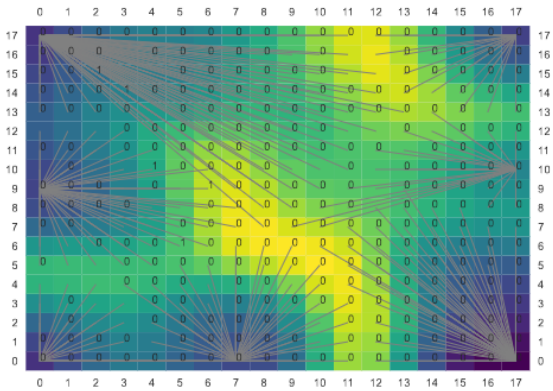
of 96.0%. When run without feature selection, the model increases its accuracy to 98.2% and 98.0% for normal and pruned NSL-KDD variants and 96.7% and 95.7% for CIC-IDS-2017. The GHSOM is able to perform better when it has more features. Because of its ability to grow both vertically and horizontally, it is able to make connections in data that its predecessors can not. We also tested the GSOM model using all features. Unlike the GHSOM, it does not benefit from having more information. Overall, the GHSOM with no feature selection performs the best out of all of our CL models.



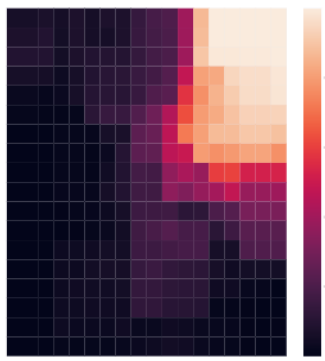
(a) NSL-KDD Starburst U-Matrix



(b) Dst byte Feature Map



(c) CIC-IDS-2017 Starburst U-Matrix



(d) Flow bytes/s Feature Map

Figure 5.6: (a)(d)The Starburst U-Matrix shows both the most common label for each node and the clusters the SOM has learned. Darker areas represent units that are close Euclidean Distance-wise. Notably, we can see a clear divide between classes on the NSL-KDD dataset as represented in the figure. (b)(e) K-means clustering can be used as a simplified view of where labels appear on the SOM. In this model's iteration, anomalous traffic is mostly grouped on the bottom of the SOM.(c) The feature value heatmap displays the value of a specific feature on each unit in the SOM. Lighter values represent units with values closer to 1, while darker values show values closer to 0. The 'dst byte' example shows that the bottom 'anomalous' cluster values higher values.

The models from our architecture can also be compared to other algorithms in the literature. Some of the best performing algorithms are variations of Deep Neural Networks (DNN) and other black box Neural Networks (NN). These black box models are able to capture complexities in data that no human or white box model could comprehend. However, a major problem these black box models have is that they are not easily explainable. Unlike the CL algorithms detailed above, information is given to these NNs and no explanation is given as to why a prediction is made. It is possible to make these models explainable by using out-of-the-box explainer modules such as SHAP or LIME. However, a major problem with using these explanation frameworks is their black box nature. Table 5.3 compares our results to some works in the literature. Note that these models either have no feature selection or a different feature selection than our own models, and the authors did not list all possible metrics. The NSL-KDD and CIC-IDS-2017 dataset models are all similar in accuracy. Our GHSOM architecture has between 1% - 3% lower accuracy compared to others, however, the GHSOM is by far more explainable. Even with the small loss in accuracy, we believe that an explainable IDS can be more beneficial for users. Using the explanations given to us by our models, it may be possible to modify parts of the architecture to match the black box models' accuracy. Additionally, the explanations are created using a white box algorithm. The user can be confident in how the explanations were generated. Therefore, they may have more confidence that the model and explanations are trustworthy.

### 5.4.2 Explanation Generation

Explanation generation can be divided into two subcategories: Statistical and Visual. Statistical explanations for our architecture consist of Global and local feature significance charts.

These explanations can help build a general understanding of the model but lack topographical information. Visual explanations are datamined from the CL models and allow the user to combine the statistical explanations with topographical information. These include the U-matrix, feature heatmap, and label map. In this section, we will discuss the GSOM's explanations. The GSOM performed just as well as the GHSOM with regard to accuracy and uses the same explanations as the GHSOM. However, the GHSOM may require the user to look at its many different hierarchical GSOMs to be useful. Therefore, any method used to understand the GSOM model can also be expanded to the GHSOM.

#### 5.4.2.1 Global Explanations

Global significance explanations allow the user to form a general understanding of models and datasets. 'Bayesian probability of significance' is the method we chose to create global explanations [33]. This algorithm calculates feature variance where, in theory, features with higher variance are more likely to cause clustering in data. Using explanations created with this method, users can begin to create a strategy for examining explanations. This can help a user when faced with a dataset with many features. However, a global explanation is not guaranteed to be useful for all predictions. This type of global explanation is probabilistic in nature. Some local predictions may use the least probable features when choosing a label. Global feature significance explanations should be used as a guide to help examine the datasets and explanatory outputs.

The global explanations for our two datasets can be found in Figures 5.5b and 5.5d. More important features are denoted with higher significance. For the NSL-KDD dataset, we can see the top three most significant features are 'Destination (dst) bytes', 'dst host count', and 'Source

(src) bytes'. Additionally, we can see that even though these three are the most important, the other features have high enough significance that they could play a role in predictions. The CIC-IDS-2017 global explanation tells a different story. 'Flow bytes/s' has the highest variance among all of the features. We can classify this as the most significant feature while viewing the next 11 features ('flow duration' to 'Forward (fwd) packets/s') as being somewhat significant. The final five features are less likely to impact the model.

Using the global explanations in Figures 5.5b and 5.5d, we can form some initial conclusions about the models and datasets. For NSL-KDD, benign and malicious packets differ in size, length, and number. This is demonstrated by 'dst host count', 'duration', and 'dst bytes'. At this stage, forming any other conclusions may not be beneficial. One of these features may be a good starting point to look at for later explanation types. Malicious and benign traffic in the CIC-IDS-2017 dataset appears to vary in how much data is sent over the network. Similar to the NSL-KDD dataset, it may not be a good idea to form a more concrete opinion about the models just yet. However, we do know that malicious and benign data differ in the ways mentioned above, and these ideas may be good concepts to investigate.

#### 5.4.2.2 Local Explanation

The local prediction explanations for GSOM anomalies can be found in Figures 5.5a and 5.5c. Using local explanations, users can take their coarse understanding of the dataset and begin fine tuning it. Here, we will look at an anomalous local explanation from each dataset, however, it may be beneficial for a user to view many at one time. Local explanations are created by examining a

feature's proximity to its BMU counterpart (see Section 5.2.4). The most important features have higher values.

The anomalous prediction for NSL-KDD in Figure 5.5a can be used to demonstrate how to understand local prediction explanations. For this anomaly, 'Destination (dst) host count', 'duration', and 'dst bytes' were the most important features. CIC-IDS-2017's local anomalous explanation has some interesting features. Eight features have a proximity of one or very near to one. Five of its features are of some significance to the prediction, while the first four are less or not significant.

To be able to better understand how the model labels predictions, multiple local explanations should be used. Ideally, the user would need to look at many anomalous and benign explanations. Doing so would allow the user to solidify their idea of how the model makes a malicious or benign prediction. Looking at our NSL-KDD anomaly explanation, it is similar to the global explanation. 'Destination (dst) host count', 'dst bytes', and 'src bytes' are all of higher importance. However, 'duration' has also made a large impact even though it was considered less significant globally. If we look at other malicious prediction explanations, we see a similar trend. The NSL-KDD model uses these features to make many anomalous predictions. The CIC-IDS-2017 explanation states that 'flow bytes/s' and seven other features are what cause this anomaly. Similar to NSL-KDD, this trend tends to hold true over many other anomalous predictions. One or more of the features may change places of importance, but a pattern can be seen in the explanations.

The user, now having examined both global and local explanations, has hopefully begun to form a mental model of how the GSOM makes predictions. The process used in this section can be used to form a general understanding of the SOM and GHSOM models. Using clues discovered in their

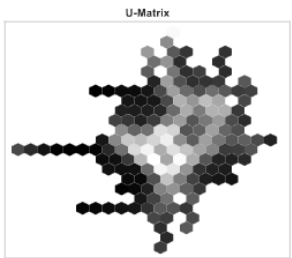
investigation, a user may want to view visual explanations for specific features. The topological information gleaned from these explanations may prove fruitful in understanding the model even better.

#### 5.4.2.3 Visual Explanations

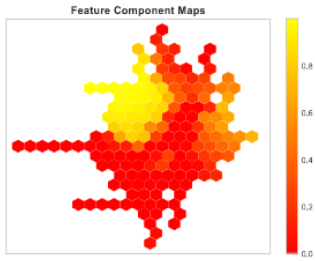
Unlike the statistical explanations, the three visual explanations need to be viewed together. Examples of these explanations can be seen in Figure 5.7. The U-matrix is a hexagonal grid composed of dark and light nodes. Darker cells represent nodes in the GSOM that are closer to one another. Lighter cells denote a separation of nodes and clusters. The label map contains the label used by each node for prediction. Light yellow nodes assign a prediction label of *malicious* while dark red nodes assign a prediction label of *benign*. A shade in between yellow and red indicate when a node has a probability of choosing either *benign* or *malicious*. Finally, the feature heatmap contains visualized information of a single feature in the GSOM. Since each node represents a sample from the dataset, we are able to visually map feature values. Lighter colored nodes contain a feature value closer to one, while the darker values are closer to zero.

Firstly, we can look at the U-matrix to see how many clusters there are and where they formed. The NSL-KDD model created about five clusters seen in Figure 5.7a. Four clusters appear on the edge of the map with a smaller one in the middle. In Figure 5.7d, the CIC-IDS-2017 model created between four to six clusters. Similar to the NSL-KDD U-matrix, the user may see that there is a cluster at the top, left, bottom, right, and middle of the map. Now that we have defined the clusters, we can look at each cluster's associated label. The NSL-KDD model appears to label the bottom-left and middle clusters as anomalous. The other clusters seem to be mostly benign.

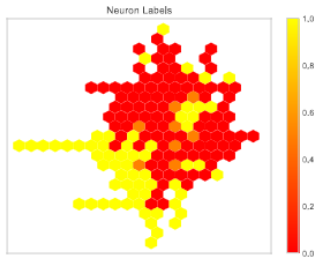
For CIC-IDS-2017, we can see that nearly the entire map is benign. There are a few clusters of malicious labels, but they are intermixed with benign data. Lastly, we can look at the meaning of the feature heat map. The NSL-KDD feature heat map represents the 'desetination (dst) bytes' feature. Here, we can see that the top-left of the map contains higher values of 'dst bytes' while the rest of the map contains much lower values. Similarly, the CIC-IDS-2017 dataset's 'flow bytes/s' is much higher in the bottom-left than anywhere else on the map.



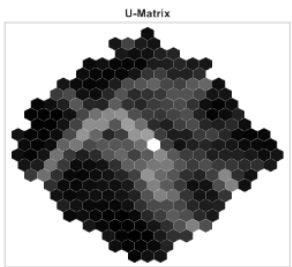

(a) NSL-KDD U-Matrix



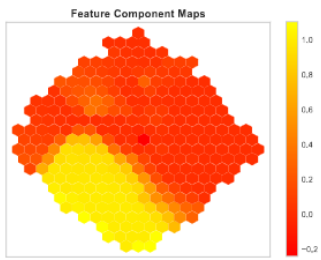

(b) Feature Component Map of Dst-Bytes



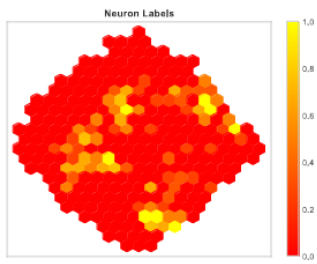

(c) NSL-KDD Neuron Labels




(d) CIC-IDS-2017 U-Matrix




(e) Feature Component Map of Flow Bytes/s




(f) CIC-IDS-2017 Neuron Labels

Figure 5.7: Visualizations generated from a GSOM for models trained on NSL-KDD and CIC-IDS-2107. (a)(d) The U-matrix maintains the same properties as the SOM starburst visualization with darker areas representing neurons closer together. (b)(e) The Feature Component Map also shares the same properties as the SOM feature map in Figure 5.6. (c)(f) The Neuron Label map shows the class label represented by a red or yellow color.

With our new understanding of visual explanations, we can combine them to form more complex ideas about the model. A user can view both the feature heat map and the U-matrix and see that there is a cluster of nodes associated with high 'dst bytes' values. They can then look to the neuron label map and see that this cluster is associated with benign data. This thought process can be used with the other features from the dataset. The user can then decide that they want to view the dataset's next most significant feature. In NSL-KDD's case, the next best feature would be 'dst host count'. Alternatively, the user could decide to use a local explanation to make decisions about which features to view. The process outlined above can be used for larger datasets such as CIC-IDS-2017 to guide the user into forming conclusions about the model.

#### 5.4.2.4 User Conclusions

Having looked at all of the explanations, a user may now make a few decisions. The first is that the explanations were sufficient such that they can complete their *task*. The second is that they believe that the model could perform better or output better explanations that could benefit their *task*. The NSL-KDD explanations may be sufficient for a user, however, the CIC-IDS-2017 model has some flaws that could affect its accuracy. The user could see that the CIC-IDS-2017 explanations heavily favor benign data. Using the architecture diagram defined in Figure 5.1, the user may modify different aspects of the architecture. In this case, the user may decide to preprocess the CIC-IDS-2017 dataset differently. It is possible that stratifying the training portion of the dataset would create a more balanced set of clusters.

#### 5.4.2.5 SOM and GHSOM Explanations

The process outlined above can also be used to understand both SOM and GHSOM explanations. In fact, the global and local explanations for these two algorithms look the same. However, the SOM algorithm that we chose uses a visual explanation that combines the U-matrix and label map. It also includes a starburst-like pattern that helps dictate where the centers of the clusters are and how far they stretch. We leave the SOM explanations in Figure 5.6 as an exercise for the reader to form their own conclusions using the above process. Lastly, the GHSOM uses the same visual explanations as the GSOM, but it would require many different images of each to explain. Instead, we have created a visualization representing the hierarchical structure of the GHSOM that includes how each GSOM within labels data. This visualization for the pruned GHSOM can be seen in Figures 5.4. In the figure, red represents a node that labels data as malicious, blue nodes are benign, and yellow indicates a branching node.

## 5.5 Conclusion

In this work, we created an Explainable Intrusion Detection (X-IDS) architecture featuring three Competitive Learning (CL) based algorithms. It was built using DARPA's recommended guidelines for an explainable system. The architecture consists of four phases: Pre-Modeling, Modeling, Post-Modeling Optimization, and Prediction Explanation. In the Pre-Modeling phase, we preprocess datasets and select our initial model parameters. In the Modeling phase, we train the Self Organizing Map (SOM), Growing Self Organizing Map (GSOM) and Growing Hierarchical Self Organizing Map (GHSOM) and record quality metrics. In the Post-Modeling Optimization phase, we find better model parameters to achieve higher accuracy results, and we implement a

pruning process for the GHSOM model. Lastly, we generate explanations and allow the user to make modifications to the architecture in the Prediction Explanation phase. When compared with existing Error Based Learning (EBL) algorithms, CL algorithms are less accurate. However, CL algorithms are far more explainable, leading to a more trustworthy IDS.

The main objective of this work was to demonstrate the explanatory properties of CL algorithms. In our explanation discussion, we showed that CL algorithms are highly explainable because of their ability to mimic patterns in data. This is a feature that EBL techniques lack. We demonstrated a strategy that a user could use to be able to understand explanations to better trust or improve the model. This strategy involved using course grain explanations such as the global and local feature significance charts to form a general understanding of the models. With this general understanding, users can then use the feature heatmaps, U-matrices, and label maps to form a more comprehensive knowledge of the model.

Additionally, a pruning process was applied on the GHSOM in an effort to lower the number of branches it generated. We were able to decrease the size of the GHSOM by 92% - 99% while only losing 0.2% - 1.0% accuracy. This lowers the performative overhead and allows the pruned GHSOM to make predictions faster than the unpruned GHSOM. Additionally, reducing the size of the GHSOM can help with visualizing explanations.

Lastly, a performance analysis was performed on our CL-based X-IDS architecture. Tests were run using the NSL-KDD and CIC-IDS-2017 datasets. Our experimental results showed that the CL models can achieve accuracies as high as 98.2% on the NSL-KDD dataset and 96.7% on the CIC-IDS-2017 dataset. We compare these results with existing EBL algorithms. We find that EBL

algorithms are 1% to 3% more accurate than the CL algorithms, however, EBL models are far less explainable.

The future for intrusion detection is explainability. Using architectures and methods, such as the ones used in this paper, will lead to more powerful and trustworthy IDS. White box methods can be improved and adapted to create more accurate AI models. Competitive Learning algorithms embody this philosophy. Their explainability, ease of use, and low performative cost allow for them to be the front runners for future X-IDS.

# CHAPTER 6

# WHITE BOX ECLECTIC RULE EXTRACTION FOR EXPLAINABLE DEEP NEURAL NETWORK IDS

## 6.1 Introduction

The ubiquity of black box algorithms and black box surrogate explainers create trust issues for Explainable Intrusion Detection Systems (X-IDS) [20]. Explainable Artificial Intelligence (XAI) was created as a means to increase the transparency of these black box approaches [95]. Historically, white box techniques were used to create explanations for black box models. More recently, the use of surrogate explainers, such as Local Interpretable Model-Agnostic Explanations (LIME) [85] or SHapley Additive exPlanations (SHAP) [60], have become more common. These techniques are used to create local and global explanations for neural networks but are themselves black boxes. By using these techniques, we take a step back in explainability. If one cannot trust a black box model because it is opaque, how can one trust a black box surrogate explainer?

One solution to this problem is the use of pedagogical, white box explanations for neural networks. Pedagogical algorithms take a similar approach to black box surrogate explainers. They use the neural network inputs and outputs to create an approximate model [9, 10]. A popular technique is to train Decision Trees (DT) as a surrogate model. Pedagogical approaches have the benefit of being fast and scalable. This method, however, is also lacking with regard to trust. Since pedagogical methods do not use the black box neural network weights, they cannot create a

trustworthy surrogate model. Decompositional Rule Extraction (RE) can alleviate this issue. By training DTs using the weights from each layer [9, 10], we can create trustworthy rules for black box neural networks. What decompositional RE gains in trustability, it loses in scalability. A major issue with this type of algorithm is its exponential scaling due to its need to stitch each layer's rules together from input to output.

Another option is to use eclectic rule extraction. Eclectic RE uses techniques from both pedagogical and decompositional algorithms [10]. Eclectic algorithms offer a middle ground between the scalability of pedagogical techniques and the trustworthiness of decompositional techniques. It trains one or more DTs for each hidden layer which are used to extract rules for a ruleset. Due to this, eclectic rule extraction scales polynomially with respect to the number of layers. This scaling issue can be mitigated using the eclectic algorithm's customizability. For larger Deep Neural Networks (DNN), one can extract rules from a subset of layers rather than all layers. Additionally, the eclectic rule extractor gains the benefit of trust from generating rulesets using weights from the black box neural network. This makes it more trustworthy than black box surrogates and pedagogical approaches.

X-IDS heavily benefits from explainability and trust [71]. Explainability allows security experts to understand how and why their IDS is making predictions. Experts can use eclectic RE as a means to understand their IDS by generating global, explainable rulesets. Using this information, security experts are able to make modifications to their IDS in order to increase its accuracy. Additionally, experts have other tasks that they need to perform to protect their systems. Having more trust in the IDS can help experts perform tasks in a more timely and confident manner. This leads to more reliable network defense.

In this work, we present a hybrid X-IDS architecture that uses white box eclectic RE to generate explanations. The proposed solution is a white box surrogate explainer that utilizes the DNN's hidden layers to generate an explainable ruleset. DNN models are trained using the UNSW-NB15 and CIC-IDS-2017 datasets, and explainable rulesets are created using the eclectic RE algorithm. We find that the RE algorithm is able to generate rulesets that mimic the models' outputs at an accuracy of 99.9%. Additionally, the rulesets have similar accuracy to the DNN models when compared to the datasets' ground truth labels.

Major contributions presented in this work are -

- A hybrid X-IDS architecture using a black box DNN predictor and a white box surrogate explainer. Eclectic RE is used to generate human-readable rules from the hidden neurons of a DNN. RE creates a global, explainable ruleset that can be used to help users understand how and why their model makes predictions.
- An eclectic rule extraction algorithm that can be run on intrusion detection datasets. This algorithm can be run for both binary and categorical predictions. An eclectic RE algorithm gives the user flexibility when determining how much of the model they would like to explain. This can increase ruleset extraction speed. Rulesets generated using this algorithm are highly similar to the DNNs' predictions.
- A performative and explanatory analysis of our architecture using modern datasets. Our model is tested against the CIC-IDS-2017 and UNSW-NB15 datasets. Using these datasets, we train and test our DNN and create accurate rulesets. Rulesets are able to mimic the DNNs' outputs with an accuracy of 99.9%. We discuss the trade-off of speed and performance and detail the rulesets' explainability.

## 6.2 X-IDS Architecture

One important goal that X-IDSs have is to aid users in understanding predictions that can aid them in certain tasks. CSoC security analysts, for example, are tasked with protecting a given network from attack. To help users such as this protect their networks, we propose a hybrid X-IDS architecture that uses a DNN to create predictions and eclectic RE to create explanations. The

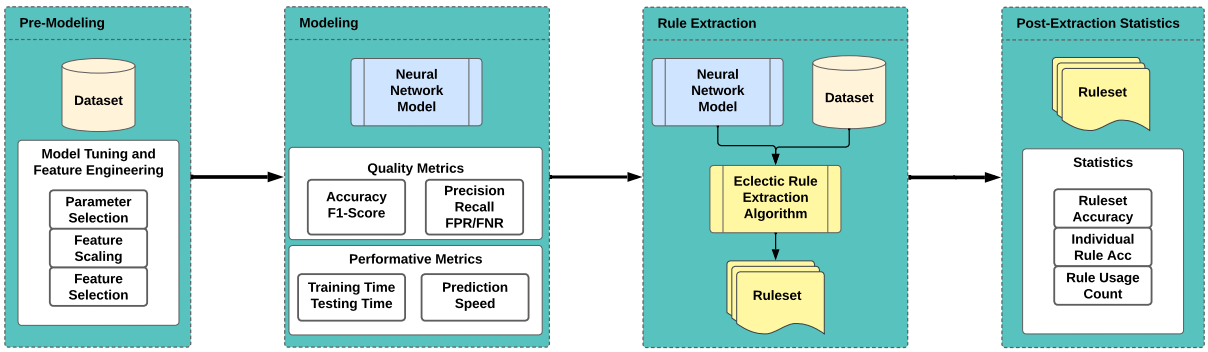



Figure 6.1: Architecture for a surrogate explainer X-IDS. It features four total phases. In the Pre-Modeling phase, the datasets are feature engineered to be compatible with the neural network and RE algorithm. Model parameters are also selected here. The model is trained and tested in the Modeling phase. Here, we record important quality and performative metrics. The trained model and dataset can then be used to extract a ruleset. Lastly, we generate statistics for the ruleset and rules to aid the user in their understanding.

architecture is divided into four phases. First, the datasets are preprocessed and model parameters are tuned in the Pre-Modeling phase. Second, the DNN are trained and various quality metrics can be recorded. Third, rules are extracted from the model to form rulesets. Fourth, the rulesets are tested for various statistical measures. The architecture diagram for our X-IDS can be found in Figure 6.1.

### 6.2.1 Pre Modeling

The first phase in the architecture is Pre Modeling. Here, we construct high-quality datasets and determine model hyper-parameters. This work uses the CIC-IDS-2017 [80] and UNSW-NB15 [69] datasets. There are a number of reasons why we chose these two datasets. First, these datasets use more modern attacks when compared to older datasets. CIC-IDS-2017 and UNSW-NB15 were developed in 2017 and 2015 respectively. Both of these datasets were created to offer 'up-to-date' attacks. Although these datasets are six years old, they can be used to give a good impression

of how our model will work with real-world data. Second, CIC-IDS-2017 contains 2.8 million samples, while UNSW-NB15 contains just over 250,000 samples. This allows us to stress-test our model by recording training testing and rule generation times for datasets that are an order of magnitude different in size. Understanding the scalability of our model and explanations is a crucial factor for intrusion detection. Additionally, we are able to create large validation and testing datasets that contain data that may not appear in the training dataset. A description of how we preprocessed the datasets can be found in Section 6.3.

### 6.2.2 Modeling

The next phase is Modeling. In this phase, we train the black box neural network and record quality and performative metrics. To construct the NN, we use Tensorflow [2]. It consists of an input layer, two hidden layers, and an output layer. The two hidden layers each contain 64 neurons and use the *ReLU* activation function. The output layer uses the *Sigmoid* activation function for binary classification. To optimize the model, we selected the *Adam* optimizer. After the model has been trained, it can be tested for quality and performative metrics. Table 6.1 shows the DNN parameters we used during training.

*Quality & Performative Metrics:* For our experiments, we record many traditional quality metrics. These include accuracy, F1-score, precision, recall, False Positive Rate (FPR), and False Negative Rate (FNR). Accuracy compares the number of true positives and negatives to the number of false positives and negatives. This gives a general idea of how well the model performs as a whole, however, its use may be misleading with imbalanced datasets. F1-score, on the other hand, accounts for this by using precision and recall to define its score. This helps to minimize the

Table 6.1: DNN Training Parameters

| **Parameter Name** | **Parameter Value** |
|---|---|
| Hidden Layers | 2 |
| Hidden Layer Neuron Count | 64 |
| Output Layer Neuron Count | 2 |
| Hidden Layer Activation Function | Rectified Linear Unit (ReLU) |
| Output Layer Activation Function | Softmax |
| Bactch Size | 64 |
| Training Iterations | 100 |
| Early Stopping (Validation Loss) | 5 |

effects of an imbalanced dataset. The last two metrics are FPR and FNR. These detail how often the model mislabels anomalous data as normal and vice-versa. These are important metrics for an IDS as they denote how often an attack goes unnoticed or a benign user is prevented from using a service. There are also performance-based metrics that are important to note for an IDS. These include training, testing, and prediction times. The speed at which an IDS can be trained and tested can be vital for a network.

### 6.2.3 Rule Extraction

In this stage, we use the trained model and the training dataset to extract rules from the model's hidden layers. We outline the eclectic rule extraction algorithm in Section 2.5. Rules generated from the hidden layer can be concatenated together to form an explainable ruleset. This ruleset can

be used by the user to understand the potential decisions the model is making when determining if a sample is benign or malicious.

The rule extraction algorithm depends on the use of a Decision Tree (DT). We chose to use the Scikit-learn Decision Tree Classifier. However, there are other DT classifiers that are available, such as C5.0, that offer varying functionality and scalability. Scikit-learn's DT offers the benefit of speed and ease of use which is the reason why we chose their implementation. Additionally, the DTs come with varying hyper-parameters that can be used to alter the training process. Notably, the 'max depth' and 'max leaves' hyper-parameters can be used to limit the size and number of rules generated. We can modify these parameters to find the optimal trade-off between speed and ruleset accuracy. Next, it is possible to modify the DT training process by changing the amount of data they train on. One of the main scalability factors this eclectic RE algorithm has is dataset size. One could decide to only use a subset of the original training dataset in order to speed up the rule extraction process. This comes with its own downsides, however. By limiting the amount of data the DTs are trained on, one may be leaving out vital information for an accurate ruleset. Lastly, one can decide to extract rules from specific DNN hidden layers. Since the RE algorithm repeats for every hidden layer, this parameter has the highest impact on RE speed.

Table 6.2: Results from the unbounded, leaves, and layers tests for the Eclectic RE algorithm

**UNSW-NB15**

| Experiment | Num. Rules | Ground Truth Accuracy | Model Prediction Accuracy | Average Terms | Longest Rule | Extraction Time (s) | Testing Time (s) | Testing Std (s) |
|---|---|---|---|---|---|---|---|---|
| **Unbounded** | 2380 | 93.6% | 99.1% | 15.8 | 30 | 1610 | 723 | 79 |
| **2000 Leaves** | 2421 | 93.6% | 99.0% | 15.7 | 30 | 1600 | 805 | 140 |
| **1000 Leaves** | 1684 | 93.7% | 99.1% | 14.7 | 25 | 1603 | 549 | 95 |
| **500 Leaves** | 946 | 93.6% | 99.0% | 13.1 | 25 | 993 | 278 | 65 |
| **100 Leaves** | 184 | 93.5% | 98.5% | 4.2 | 14 | 212 | 51 | 7 |
| **10 Leaves** | 19 | 90.9% | 94.7% | 4.2 | 5 | 60 | 6.6 | .5 |
| **20 Layers** | 2278 | 93.6% | 99.0% | 20 | 20 | 1532 | 606 | 98 |
| **10 Layers** | 502 | 93.4% | 98.4% | 10 | 10 | 430 | 89 | 10 |
| **5 Layers** | 38 | 91.0% | 95.1% | 5 | 5 | 77 | 11 | 2 |

**CIC-IDS-2017**

| Experiment | Num. Rules | Ground Truth Accuracy | Model Prediction Accuracy | Average Terms | Longest Rule | Extraction Time (s) | Testing Time (s) | Testing Std (s) |
|---|---|---|---|---|---|---|---|---|
| **Unbounded** | 1686 | 93.1% | 99.9% | 14.2 | 27 | 9504 | 6882 | 1321 |
| **2000 Leaves** | 1815 | 93.1% | 99.9% | 14.1 | 27 | 9782 | 6211 | 566 |
| **1000 Leaves** | 1701 | 93.1% | 99.9% | 14.3 | 27 | 8969 | 4950 | 879 |
| **500 Leaves** | 1000 | 93.1% | 99.9% | 12.2 | 25 | 7276 | 3305 | 719 |
| **100 Leaves** | 188 | 93.0% | 99.8% | 8.6 | 18 | 1792 | 554 | 129 |
| **10 Leaves** | 20 | 91.5% | 97.6% | 4.7 | 6 | 664 | 67 | 11 |
| **20 Layers** | 1675 | 93.1% | 99.9% | 13.2 | 20 | 4401 | 4659 | 1409 |
| **10 Layers** | 601 | 93.0% | 99.6% | 9.1 | 10 | 2060 | 1242 | 301 |
| **5 Layers** | 57 | 91.0% | 97.0% | 4.9 | 5 | 760 | 113 | 30 |

There are some additional notes that can be made about our specific implementation. First, the algorithm is designed to work with multiclass datasets. This means that NNs need to have at least two output neurons. Multiclass classification can abstractly predict binary classes by using two output neurons and the softmax activation function. The user will need to One Hot Encode their binary class dataset. Secondly, we did not implement multiprocessing. The bottleneck for our implementation is Python's default single thread. Implementing the ability to use more than one CPU core can increase the speed of the algorithm.

### 6.2.4 Post-Extraction Statistics

Finally, we can compare the dataset and model predictions to the ruleset to obtain useful statistical data. Useful information includes ruleset accuracy, individual rule accuracy, and rule usage. Using these statistics, we can aid the user in understanding the ruleset and model. The first step in this process is ruleset evaluation.

There are some major design decisions that can be made when evaluating a ruleset. First, one can opt to take a comprehensive or greedy approach to evaluating the ruleset. We opted to use a greedy rule comparison approach. The difference between these approaches is their stopping criteria for each testing sample. In the comprehensive approach, each sample is compared to all rules. Since there is a potential for collision, an additional step would need to be made to determine which rule is more accurate. The second method is a greedy approach. Rather than compare a sample to all possible rules, we stop as soon as we find a matching rule. The benefit of this approach is that it speeds up the evaluation on average by a factor of 2. This is due to the fact that

on average a sample should only need to be tested against half the rules. This, however, does not change the potential maximum runtime.

A potential problem with RE is the number of rules it extracts. Some extraction algorithms can extract 10,000 rules. Our algorithm has extracted close to 2400 rules. These numbers are not feasible an amount for a human to comb through. To mitigate this, we can assign a usage counter and accuracy to each rule. During ruleset evaluation, these statistics can be saved in order to assist the user in understanding the ruleset. Higher used and higher accuracy rules can be used to understand the general composition of benign or malicious samples.

## 6.3 Experimental Design

The X-IDS architecture is evaluated for accuracy and speed. There are two types of accuracy evaluations used to determine the RE's effectiveness. Ground truth accuracy is the ruleset's accuracy when compared to the testing dataset's true labels. This accuracy should be compared to the DNN's testing accuracy. Model prediction accuracy is the ruleset's accuracy when compared to the DNN's outputs. We define high accuracy for ground truth accuracy as within 1% of the model's accuracy. Model prediction accuracy should be greater than 99% to be considered highly accurate.

### 6.3.1 Model Parameters & Dataset Preprocessing

The DNN's were trained up to 100 epochs with a batch size of 64 and early stopping criteria. Training ends early when the validation loss does not improve over 5 epochs. Increasing the patience value past 5 did not yield better results only slower training times. In general, the models trained between 25 to 30 epochs before stopping early. The DNN models used a total of 2 hidden

layers with 64 neurons each. The output layer consisted of 2 neurons. The hidden layers used the ReLU activation function while the outputs layer used the softmax function.

The datasets used for these experiments were CIC-IDS-2017 and UNSW-NB15. Both of the datasets are preprocessed as follows. First, samples with 'NaN' values are removed. Second, categorical features are One Hot Encoded. Third, the dataset is normalized. Fourth, the binary labels are One Hot Encoded. This is necessary so that the RE algorithm is able to label rules. The dataset is then split into 60% training, 20% validation, and 20% testing datasets using Scikit-Learn's **train_test_split()** function twice. Once to split the training from the validation and testing, and once to split the validation from testing.

### 6.3.2 Rule Extraction Parameter Experiments

Experiments are run by modifying individual parameters in the RE algorithm. These parameters determine the number of layers or leaves the DTs are allowed to generate. They can also modify how much of the dataset is used to train the DTs or how many of the hidden layers rules are extracted from. The limited leaves experiments are divided into 2000, 1000, 500, 100, and 10 total leaves. Typically, this limits the total rules generated. The limited layers experiments are divided into 20, 10, and 5 layers. This parameter limits both how many rules are generated and how large they can be. The next two experiments use subsets of the dataset and hidden layers. The DTs are trained using 80%, 60%, 40%, and 20% of the training dataset. The hidden layer experiments extract rules from the first and second hidden layers and then tests them independently. All experiments are compared to rules extracted using default DT parameters. DTs trained this way are allowed to generate as many leaves and layers as they need.

Table 6.3: Results from the unbounded, dataset subset, and limited hidden layer tests for the Eclectic RE algorithm

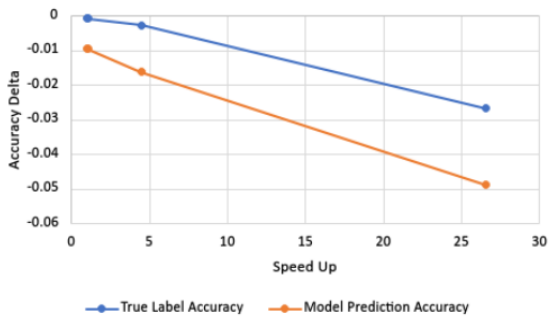
| UNSW-NB15 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Experiment** | **Num. Rules** | **Ground Truth Accuracy** | **Model Prediction Accuracy** | **Average Terms** | **Longest Rule** | **Extraction Time (s)** | **Testing Time (s)** | **Testing Std (s)** |
| **Unbounded** | 2380 | 93.6% | 99.1% | 15.8 | 30 | 1610 | 723 | 79 |
| **80% Dataset** | 2029 | 93.6% | 98.9% | 15.6 | 28 | 1106 | 535 | 44 |
| **60% Dataset** | 1735 | 93.6% | 98.8% | 14.9 | 28 | 695 | 589 | 89 |
| **40% Dataset** | 1377 | 93.4% | 98.6% | 14.4 | 27 | 358 | 324 | 23 |
| **20% Dataset** | 841 | 93.3% | 98.3% | 13.2 | 23 | 110 | 281 | 76 |
| **First Hidden** | 1391 | 93.6% | 99.0% | 15.5 | 29 | 768 | 561 | 93 |
| **Second Hidden** | 1396 | 93.6% | 99.0% | 15.5 | 30 | 828 | 471 | 63 |
| **CIC-IDS-2017** | | | | | | | | |
| **Experiment** | **Num. Rules** | **Ground Truth Accuracy** | **Model Prediction Accuracy** | **Average Terms** | **Longest Rule** | **Extraction Time (s)** | **Testing Time (s)** | **Testing Std (s)** |
| **Unbounded** | 1686 | 93.1% | 99.9% | 14.2 | 27 | 9504 | 6882 | 1321 |
| **80% Dataset** | 1474 | 93.1% | 99.9% | 14.2 | 25 | 6964 | 4388 | 1599 |
| **60% Dataset** | 1389 | 93.1% | 99.9% | 13.5 | 25 | 4244 | 4630 | 1573 |
| **40% Dataset** | 990 | 93.1% | 99.9% | 13.1 | 24 | 2416 | 3208 | 1348 |
| **20% Dataset** | 826 | 93.1% | 99.9% | 12.1 | 23 | 877 | 2295 | 753 |
| **First Hidden** | 905 | 93.1% | 99.9% | 14.1 | 27 | 4625 | 4184 | 1138 |
| **Second Hidden** | 909 | 93.1% | 99.9% | 14.1 | 27 | 4837 | 3509 | 636 |

### 6.3.3 Explainability Discussion

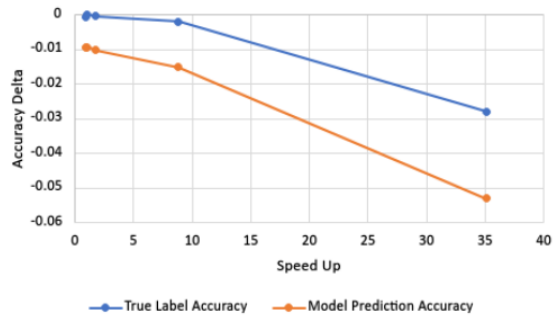
Explainability is difficult to measure. Rather, we discuss the feasibility that one could use a ruleset to understand the DNN. This is done using the total number of rules generated, their average and max length, and the model prediction accuracy. Additionally, we discuss if using a subset of the hidden layers to extract rules reduces explainability and trust.

## 6.4 Experiment Results & Evaluation

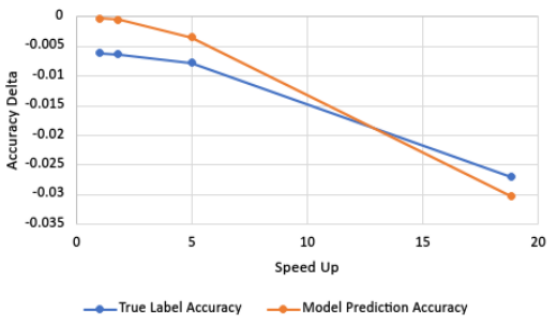
This section is divided into the different experiments outlined in the previous section. The UNSW-NB15 DNN model had an accuracy of 93.7% and the CIC-IDS-2017 DNN model had an accuracy of 93.1%. These results can be compared to the rulesets' ground truth prediction accuracies. All results for the above-outlined experiments can be found in Tables 6.2 and 6.3.



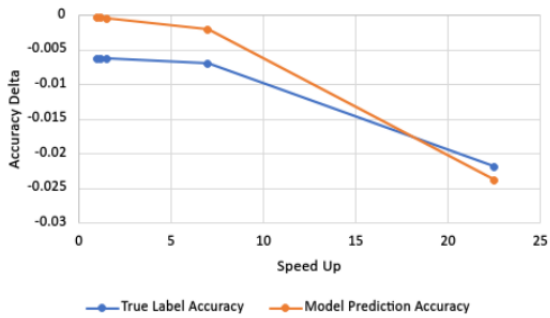

(a) UNSW-NB15 layer speed up compared to accuracy loss.




(b) UNSW-NB15 leaves speed up compared to accuracy loss.




(c) CIC-IDS-2017 layer speed up compared to accuracy loss.




(d) CIC-IDS-2017 leaves speed up compared to accuracy loss.

Figure 6.2: These charts compare the speed up versus accuracy loss for the UNSW-NB15 and CIC-IDS-2017 rulesets. True label accuracy is the rulesets label versus the testing datasets labels. Model prediction accuracy is the rulesets labels versus the models' predicted outputs.

### 6.4.1 Unbounded Eclectic Rule Extraction

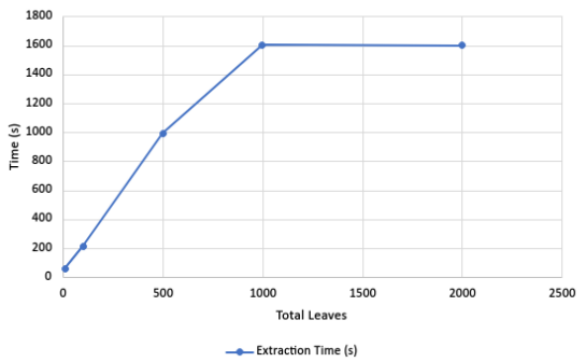
The first experiment used default DT parameters and the full training dataset. This allows the DTs to theoretically use an infinite number of leaves and layers. Rulesets generated this way were nearly as accurate as their DNN and had high model prediction accuracy. However, a major downside to creating rulesets this way is the amount of time needed and the size of the rules. The UNSW-NB15 model took 1610 seconds to extract and, on average, 723 seconds to test. Testing had a standard deviation of 78.6 seconds. UNSW had a model prediction accuracy of 99.04% and a ground truth accuracy of 93.6%. CIC-IDS-2017 was extracted in 9504 seconds (2.6 hours) and tested in 6962 seconds (1.9 hours). This model's testing speed had a standard deviation of 1511 seconds (.4 hours). It had a model prediction accuracy of 99.96% and a ground truth accuracy of 93.1%. Both datasets' prediction accuracy and true label accuracy are within one percentage point, which is considered highly accurate for this study. As one can see, there is a large deviation in testing times. There are several potential reasons behind this. First, the ruleset can accurately separate data so that few samples have overlapping rules. This can mean samples need to be tested against many rules before they find a match. Additionally, some rules tend to be champions for each label. The order in which these rules are placed in the dataset can drastically change the speed the algorithm runs. Lastly, longer rules that are checked before finding the matching rule will also increase runtime.

### 6.4.2 Limited Leaves

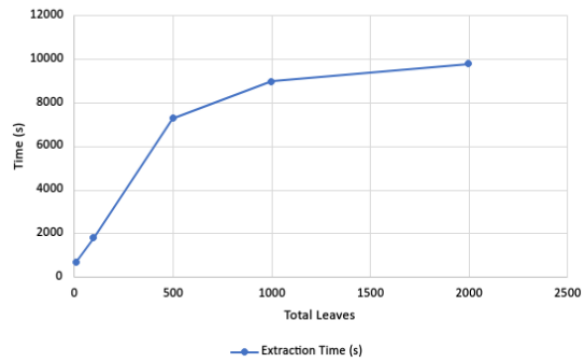
The next set of experiments limits the number of leaves that DTs are allowed to generate. This limits the rulesets in a few ways. First, there is a maximum amount of rules that are allowed to

be generated. This number is $\leq n * 2$, where $n$ is the limit of leaves. This phenomenon can be seen in Figure 6.3. For both datasets, the 500 leaves and below are only able to generate less than or equal to the maximum allowed rules. Some of these experiments do not reach their maximum due to the second hidden layer creating duplicate rules. Second, although there is no explicit limit on the number of layers, limiting the leaves can limit the number of layers. This effect can be seen in the 100 and 10 leaves limited experiments. We see that accuracy is associated with the number of rules, however, there is an upper limit on the number of rules needed for high accuracy. Users are able to limit the RE algorithm greatly before accuracy begins to degrade below our 1% criteria. UNSW-NB15 can be limited to 500 leaves and still maintain high accuracy. Its speed can be increased further if only 100 leaves are used, but its ability to mimic model output degrades by 0.5%. CIC-IDS-2017 is able to maintain high accuracy when limited to 100 leaves. Likely, this is due to the large training dataset size.

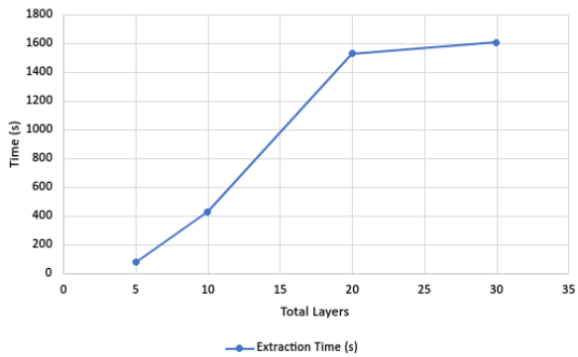
Figure 6.2 demonstrates the trade-off of accuracy to speed. We define speedup as the combination of unbounded training and average testing time divided by the limited experiment's training and average testing time. Depending on the dataset, one can see 5 to 10 times speedup before losing 1% accuracy. Extraction times are visualized in Figure 6.3a and 6.3b. Here we can see that the algorithm scales almost logarithmically with respect to the number of leaves. Figures 6.4a and 6.4b illustrate how the number of rules grows with the limited leaves tests. These graphs have a similar trend to the extraction speed graphs. We see an almost logarithmic increase in rules or a linear increase with a plateau.



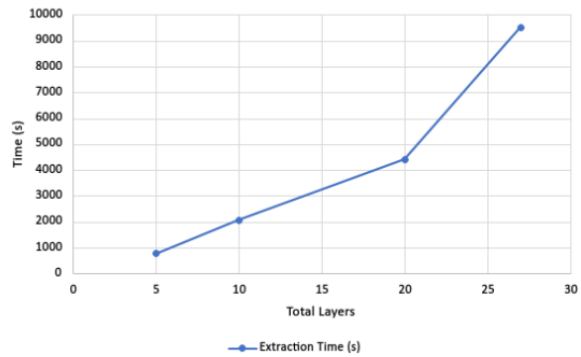

(a) UNSW-NB15 limited leaves test extraction speed.




(b) CIC-IDS-2017 limited leaves test extraction speed.




(c) UNSW-NB15 limited layers test extraction speed.

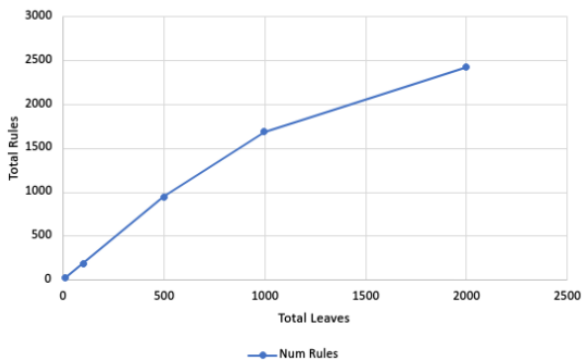



(d) CIC-IDS-2017 limited layers test extraction speed.

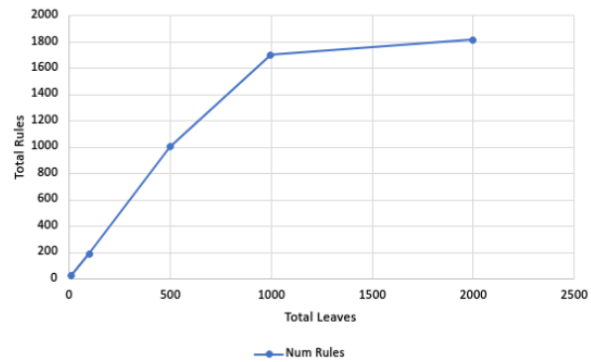
Figure 6.3: These charts show the extraction speed comparison between the various tests. Most of the results demonstrate a logarithmic scale. The outlier in Figure 6.3d is likely due to the greedy labeling process used to train the second decision tree during the extraction algorithm.

### 6.4.3 Limited Layers

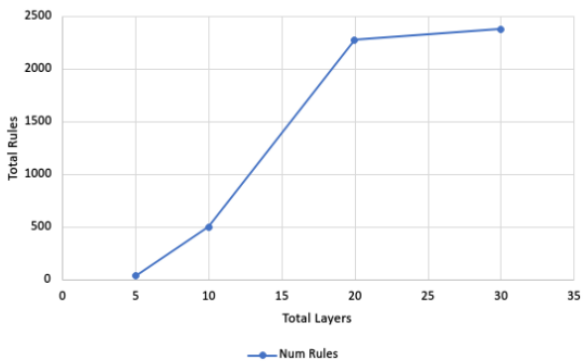
Table 6.2 also shows the results from the limited layers test, and Figures 6.3c,d and 6.4c,d visualize the results. Limiting the lasyers strictly affect the number of layers and implicitly restricts the number of rules. These experiments demonstrate the ability to speed up the algorithms by limiting ruleset creation. The total number of rules for the unbounded and 20 layer experiments are similar. However, the extraction time is reduced by a factor of 2. This is likely due to the average length and longest rules being smaller. UNSW-NB15 is able to maintain high accuracy using the 20 layer limitation. It loses 0.6% model prediction accuracy when limited to 10 layers. CIC-IDS-2017 follows a similar trend the the previous experiment. It is able to have high accuracy even with the more limited parameters. Even with only 5 layers, it is able to mimic the model's predictions with an accuracy of 97%. Again, its ability to maintain high accuracy when compared to UNSW-NB15 is likely due to the larger amount of training samples.



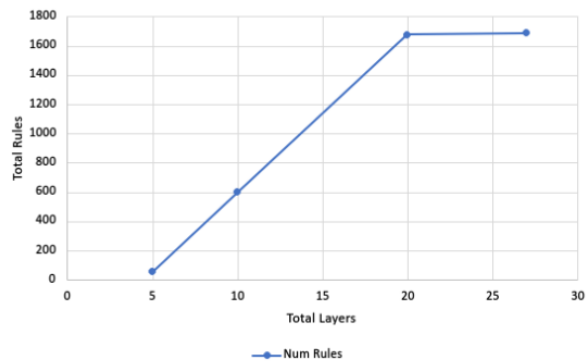

(a) UNSW-NB15 limited leaves test compared to the number of rules generated.




(b) CIC-IDS-2017 limited leaves test compared to the number of rules generated.




(c) UNSW-NB15 limited layers test compared to the number of rules generated.

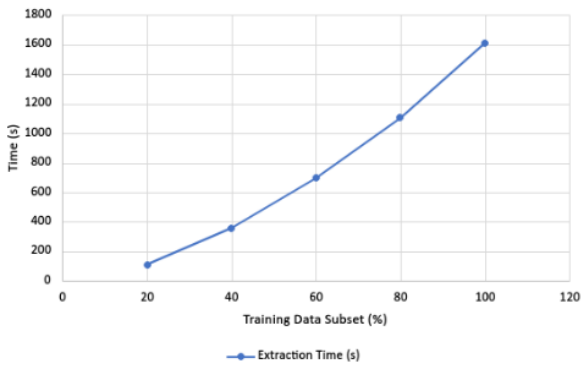



(d) CIC-IDS-2017 limited layers test compared to the number of rules generated.

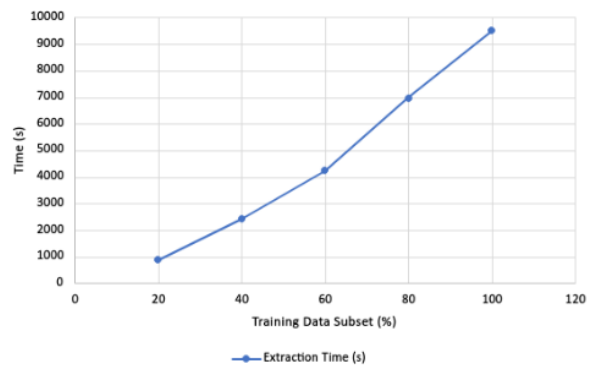
Figure 6.4: These charts demonstrate how the number of rules generated scales with the total number of leaves and layers. Limiting the decision trees to a certain number of leaves shows a logarithmic increase in rule generation. When limiting the total number of layers, we see a linear increase until a plateau.

### 6.4.4 Training Data Subsets

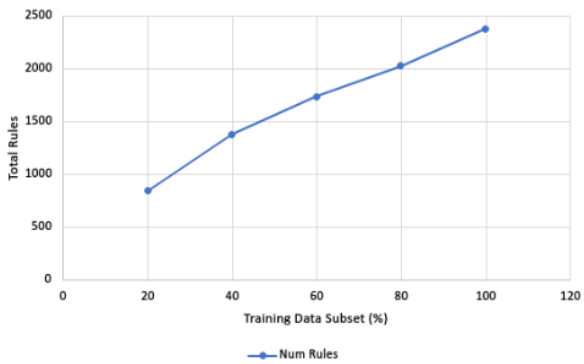
The training data subset experiments seek to improve speed by limiting the amount of training data used to create rulesets. The results for this experiment can be found in Table 6.3. Generally, we see linear increases in extraction and testing time with respect to dataset size (see Figures 6.5). Here we see that training dataset size is an important factor for model prediction accuracy. Although minor, we see an immediate degradation of model prediction accuracy for UNSW-NB15. Ground truth accuracy is able to maintain high accuracy, but we begin to lose model explainability. On the other hand, CIC-IDS-2017 is able to maintain high accuracy throughout all the subset experiments. 20% of the CIC-IDS-2017 dataset is still larger than the UNSW-NB15 dataset.



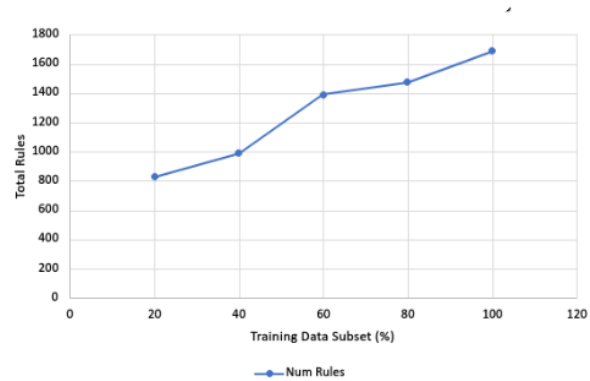

(a) UNSW-NB15 training data subset extraction speed.




(b) CIC-IDS-2017 training data subset extraction speed.




(c) UNSW-NB15 training data subset to number of rules generated.




(d) CIC-IDS-2017 training data subset to number of rules generated.

Figure 6.5: These charts illustrate the results from the training data subset experiments. The results include the total time the extraction algorithm took to extract rules and the total number of rules generated.

### 6.4.5 Limited DNN Hidden Layers

The last set of experiments tests how rulesets generated from each layer perform. These results can be seen in Table 6.3. Using this parameter cuts the number of rules in half and greatly increases the algorithm's speed. However, using this, debatably, limits the ruleset's explainability. Rather than explaining the full model, one is only explaining part of the model. In our case, we are only explaining half of the model. UNSW-NB15 is able to maintain 99% model prediction accuracy and have similar ground truth accuracy to the DNN. Notably, we see that model prediction accuracy is reduced by .1% when compared to the unbounded ruleset. This could mean that the second DNN layer is likely to produce rules that overlap with the first hidden layer. With the CIC-IDS-2017 rulesets, we see that they maintain 99.9% model prediction accuracy for both hidden layers, and they keep the same ground truth accuracy as the unbounded tests. Adding more weight to the idea that the layers produce similar rulesets. This could be an argument against the idea that using fewer layers means less explainability.

### 6.4.6 Explainability Discussion

Due to the size of some of the rulesets, it is important to discuss the usability of eclectic rule extraction. Additionally, it is important to discuss the explainability and trustworthiness of certain limited rulesets. Our RE algorithm created as many as 2400 rules when unbounded. Additionally, the unbounded rulesets generated rules with an average of 15.8 terms and a max of 30 terms. These two facts combine to make it a difficult task for humans to parse rulesets. By limiting the algorithm in the various ways above, we are able to decrease the size of the rules and rulesets. This makes the rulesets easier for users to parse but potentially lowers the ruleset's accuracy. With this in mind,

we should ask a few questions. First, “does limiting the DTs decrease the ruleset’s explainability and trustworthiness?" Second, “is model prediction accuracy directly related to explainability and trustworthiness?" Third, "what methods can users use to understand rulesets?"

The first and second questions are interlinked. The answers to these questions are likely subjective and open to debate. One user may value model prediction accuracy and ground truth accuracy similarity over all other metrics. This is because they are the only concrete statistics that one can use to compare DNN model and ruleset. Limiting ruleset creation would only decrease explainability when accuracy begins to degrade. The question then becomes “How much can accuracy degrade before a user begins to lose trust in the ruleset?" Another user may value information as a means of determining trustworthiness. Longer rules and rulesets may seem more explainable, especially because these typically correlate with higher accuracy.

The third question can have a more concrete answer. Rulesets are able to record how many and how accurate they are with the testing dataset. Rules can then be sorted by the most used or the most correct. This is applicable for both ground truth and model prediction accuracy. Users can view the most used rules and their labels. These rules can be used to form a general, global understanding of the DNN model. Users may be able to determine which features allow for higher accuracy. Using this information, the user may be able to determine which features should be removed from the dataset to make more accurate predictions. Additionally, one can use the how DT algorithms train to their advantage. Scikit-learn’s DT mainly focuses on information gain. This means that higher-level terms will typically have more variance. These terms will appear in more rules meaning they are more significant than other terms and features. Lastly, it may be possible

to use an algorithm to summarize the rulesets. This could be useful on larger rulesets, but it may run into the issue of explaining a white box with a black box.

### 6.5 Conclusion

In this paper, we created an X-IDS architecture that uses eclectic rule extraction to generate explanations for a DNN. Our X-IDS created rulesets that were 99.9% accurate when compared to our models' outputs. Our rulesets also had a similar accuracy to the DNN models when compared to the testing datasets true labels. The experiments run show the scalability and customizability that eclectic rule extraction algorithms have. By limiting our rule extraction algorithm, we can greatly increase its speed. However, its accuracy can begin to suffer when it is limited too much. This gives the user the choice between accuracy, explainability, and speed. Potential future works include extending the eclectic rule extraction algorithm to recurrent neural networks or other highly accurate models. For X-IDS architecture to be trusted, both the model and the explainer need to be accurate. Another potential future work could involve translating extracted rules into directly useful firewall rules. Rather than giving the user a set of rules, the rulesets themselves could be explained by creating firewall rules.

# CHAPTER 7

# CONCLUSION AND FUTURE WORK

In this dissertation, three X-IDS were created using white box techniques. These X-IDS were created to show that (i) certain white box algorithms are powerful enough to compete against black box algorithms, (ii) white box algorithms are far more explainable and trustworthy than black box algorithms, (iii) explainer modules need to be trustworthy in order to create useful explanations. First, an X-IDS based on Self Organizing Maps was created as a proof-of-concept. This X-IDS was able to produce visual and statistical explanations that were used to explain the model. Local, global, visual, and statistical explanations could be used together to form a more concrete understanding of the SOM's thought process. Second, an extended X-IDS was created that featured the SOM family of Competitive Learning algorithms. In that work, the Growing Self Organizing Map and Growing Hierarchical Self Organizing Map were used to create more accurate predictions and detailed explanations. The GSOM and GHSOM have accuracies that are comparable to other Error Based Learning models due to their increased complexity. Third, a hybrid X-IDS using a black box model and white box explainer was created. The model was a black box Deep Neural Network that was explained by an eclectic rule extraction algorithm. The eclectic RE algorithm was able to produce highly accurate rulesets with respect to the DNNs' outputs. This leads to a more trustworthy explainer and X-IDS.

## 7.1 Improving the Explainability of the GHSOM

There are many avenues for future work from this dissertation's contributions. Firstly, the size of the GHSOM makes them difficult them difficult to understand. Although explainable, a user would not be able to browse the 17000 GSOMs that our CIC-IDS-2017 GHSOM model produced. We were able to reduce the size of this model by 99.2% using a pruning algorithm. However, 120 GSOMs may still be too many for some users. A potential future work would be to create an algorithm that can summarize GSOMs textually, statistically, or visually. Similar to the RE ruleset rule usage, GHSOMs could have their individual GSOMs record individual accuracies and usage. This algorithm would ideally be a trustable white box implementation.

## 7.2 Connecting Rules to the Real World

The RE algorithm extracts textual rules that can be used to understand the model's reasoning for making predictions. However, thousands of rules of varying lengths are produced, which can be difficult for users to parse. In the RE contribution, we use ruleset accuracy and usage as a means to sort the rules. Sorting the rules allows the user to only browse the most used and most accurate rules. However, these rules have little connection to the real world. Users may want to be able to implement firewall rules using these rulesets or have a proper textual explanation. There are two potential future works for RE. First, design an algorithm that can create firewall rules that can accurately protect a network. Since an X-IDS's goal is to aid experts in completing tasks, creating recommendations for firewall rules could greatly increase the speed at which these experts operate. Secondly, design an algorithm that can take rules generated by RE and create descriptions that are easy for humans to understand. A likely candidate for this type of problem is Large Language

Models (LLMs). Although these models are black box, their ability to create human-like sentences can make them a vital tool for X-IDS.

REFERENCES


[1] "Self-Organizing Map Convergence," *Int. J. Serv. Sci. Manag. Eng. Technol.*, vol. 9, 4 2018, pp. 61–84.

[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, Software available from tensorflow.org.

[3] M. Abdel-Basset, H. Hawash, R. K. Chakrabortty, and M. J. Ryan, "Semi-supervised spatiotemporal deep learning for intrusions detection in IoT networks," *IEEE Internet of Things Journal*, vol. 8, no. 15, 2021, pp. 12251–12265.

[4] J. Ables, T. Kirby, W. Anderson, S. Mittal, S. Rahimi, I. Banicescu, and M. Seale, "Creating an Explainable Intrusion Detection System Using Self Organizing Maps," *IEEE Symposium on Computational Intelligence in Cyber Security*, 2022.

[5] D. Alahakoon, S. Halgamuge, and B. Srinivasan, "A self-growing cluster development approach to data mining," *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, 1998, vol. 3, pp. 2901–2906 vol.3.

[6] S. Albayrak, C. Scheel, D. Milosevic, and A. Muller, "Combining self-organizing map algorithms for robust and scalable intrusion detection," *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*. IEEE, 2005, vol. 2, pp. 123–130.

[7] S. Almutlaq, A. Derhab, M. M. Hassan, and K. Kaur, "Two-stage intrusion detection system in intelligent transportation systems using rule extraction methods from deep neural networks," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[8] K. Amarasinghe, K. Kenney, and M. Manic, "Toward explainable deep neural network based anomaly detection," *2018 11th International Conference on Human System Interaction (HSI)*. IEEE, 2018, pp. 311–317.

[9] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-based systems*, vol. 8, no. 6, 1995, pp. 373–389.

[10] M. G. Augasta and T. Kathirvalavakumar, "Rule extraction from neural networks—A comparative study," *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*. IEEE, 2012, pp. 404–408.

[11] R. G. Bace, P. Mell, et al., "Intrusion detection systems,", 2001.

[12] M. Bahrololum and M. Khaleghi, "Anomaly intrusion detection system using Gaussian mixture model," *2008 Third International Conference on Convergence and Hybrid Information Technology*. IEEE, 2008, vol. 1, pp. 1162–1167.

[13] M. Belouch, S. El Hadaj, and M. Idhammad, "Performance evaluation of intrusion detection based on machine learning using Apache Spark," *Procedia Computer Science*, vol. 127, 2018, pp. 1–6.

[14] A. Binder, G. Montavon, S. Lapuschkin, K.-R. Müller, and W. Samek, "Layer-wise relevance propagation for neural networks with local renormalization layers," *International Conference on Artificial Neural Networks*. Springer, 2016, pp. 63–71.

[15] M. Bitaab and S. Hashemi, "Hybrid intrusion detection: Combining decision tree and gaussian mixture model," *2017 14th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*. IEEE, 2017, pp. 8–12.

[16] G. Breard, "Evaluating Self-Organizing Map Quality Measures as Convergence Criteria," 2017.

[17] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, 2015, pp. 1153–1176.

[18] M. Carvalho, J. DeMott, R. Ford, and D. A. Wheeler, "Heartbleed 101," *IEEE security & privacy*, vol. 12, no. 4, 2014, pp. 63–67.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, 2009, pp. 15:1–15:58.

[20] K. K. Chennam, S. Mudrakola, V. U. Maheswari, R. Aluvalu, and K. G. Rao, "Black Box Models for eXplainable Artificial Intelligence," *Explainable AI: Foundations, Methodologies and Applications*, vol. 232, 2022, p. 1.

[21] Y. E. Chun, S. B. Kim, J. Y. Lee, and J. H. Woo, "Study on credit rating model using explainable AI," *The Korean Data & Information Science Society*, vol. 32, no. 2, 2021, pp. 283–295.

[22] DARPA, "Broad agency announcement explainable artificial intelligence (XAI)," *DARPA-BAA-16-53*, 2016, pp. 7–8.

[23] E. De la Hoz, A. Ortiz García, J. Ortega Lopera, E. M. De La Hoz Correa, and F. E. Mendoza Palechor, "Implementation of an intrusion detection system based on self organizing map," 2015.

[24] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, , no. 2, 1987, pp. 222–232.

[25] T. Dias, N. Oliveira, N. Sousa, I. Praça, and O. Sousa, "A Hybrid Approach for an Interpretable and Explainable Intrusion Detection System," *arXiv preprint arXiv:2111.10280*, 2021.

[26] M. Dittenbach, D. Merkl, and A. Rauber, "The growing hierarchical self-organizing map," *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. IEEE, 2000, vol. 6, pp. 15–19.

[27] B. Fritzke, "Growing grid—a self-organizing network with constant neighborhood range and adaptation strength," *Neural processing letters*, vol. 2, no. 5, 1995, pp. 9–13.

[28] D. Gunning and D. Aha, "DARPA's explainable artificial intelligence (XAI) program," *AI Magazine*, vol. 40, no. 2, 2019, pp. 44–58.

[29] S. M. Guthikonda, "Kohonen self-organizing maps," *Wittenberg University*, vol. 98, 2005.

[30] T. Hailesilassie, "Rule extraction algorithm for deep neural networks: A review," *arXiv preprint arXiv:1610.05267*, 2016.

[31] A. H. Halbouni, T. S. Gunawan, M. Halbouni, F. A. A. Assaig, M. R. Effendi, and N. Ismail, "CNN-IDS: Convolutional Neural Network for Network Intrusion Detection System," *2022 8th International Conference on Wireless and Telematics (ICWT)*. IEEE, 2022, pp. 1–4.

[32] L. Hamel, "SOM quality measures: An efficient statistical approach," 2016, vol. 428, pp. 49–59, Springer Verlag.

[33] L. Hamel and C. Brown, "Bayesian Probability Approach to Feature Significance for Infrared Spectra of Bacteria," *Applied Spectroscopy*, vol. 66, 1 2012, pp. 48–59.

[34] L. Hamel and C. Brown, "Improved Interpretability of the Unified Distance Matrix with Connected Components," *7th International Conference on Data Mining (DMIN'11)*, 4 2012.

[35] M. Hammad, N. Hewahi, and W. Elmedany, "MMM-RF: A Novel High Accuracy Multinomial Mixture Model for Network Intrusion Detection Systems," *Computers & Security*, 2022, p. 102777.

[36] M. Han and J. Kim, "Joint banknote recognition and counterfeit detection using explainable artificial intelligence," *Sensors*, vol. 19, no. 16, 2019, p. 3607.

[37] A. Holzinger, C. Biemann, C. S. Pattichis, and D. B. Kell, "What do we need to build explainable AI systems for the medical domain?," *arXiv preprint arXiv:1712.09923*, 2017.

[38] S. Iannucci, J. Ables, W. Anderson, B. Abburi, V. Cardellini, and I. Banicescu, "A Performance-Oriented Comparison of Neural Network Approaches for Anomaly-based Intrusion Detection," *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 1–7.

[39] D. Ippoliti and X. Zhou, "A-GHSOM: An adaptive growing hierarchical self organizing map for network anomaly detection," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, 2012, pp. 1576–1590.

[40] S. R. Islam, W. Eberle, S. K. Ghafoor, A. Siraj, and M. Rogers, "Domain knowledge aided explainable artificial intelligence for intrusion detection and response," *arXiv preprint arXiv:1911.09853*, 2019.

[41] Y. Jia, M. Wang, and Y. Wang, "Network intrusion detection algorithm based on deep neural network," *IET Information Security*, vol. 13, no. 1, 2019, pp. 48–53.

[42] A. Jobin, M. Ienca, and E. Vayena, "The global landscape of AI ethics guidelines," *Nature machine intelligence*, vol. 1, no. 9, 2019, pp. 389–399.

[43] E. Jussupow, K. Spohrer, A. Heinzl, and J. Gawlitza, "Augmenting medical diagnosis decisions? An investigation into physicians' decision-making process with artificial intelligence," *Information Systems Research*, vol. 32, no. 3, 2021, pp. 713–735.

[44] A. S. Khan, Z. Ahmad, J. Abdullah, and F. Ahmad, "A spectrogram image-based network anomaly detection system using deep convolutional neural network," *IEEE Access*, vol. 9, 2021, pp. 87079–87093.

[45] I. A. Khan, N. Moustafa, D. Pi, K. M. Sallam, A. Y. Zomaya, and B. Li, "A New Explainable Deep Learning Framework for Cyber Threat Discovery in Industrial IoT Networks," *IEEE Internet of Things Journal*, 2021.

[46] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, 2019, pp. 1–22.

[47] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, 1982, pp. 59–69.

[48] T. Kohonen, "Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map," *Biological cybernetics*, vol. 75, no. 4, 1996, pp. 281–291.

[49] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, 1998, pp. 1–6.

[50] T. Kohonen and T. Honkela, "Kohonen network," *Scholarpedia*, vol. 2, no. 1, 2007, p. 1568.

[51] A. P. Kuruvila, X. Meng, S. Kundu, G. Pandey, and K. Basu, "Explainable Machine Learning for Intrusion Detection via Hardware Performance Counters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[52] J. Lampinen and E. Oja, "Clustering properties of hierarchical self-organizing maps," *Journal of Mathematical Imaging and Vision*, vol. 2, 1992, pp. 261–272.

[53] C. Langin, M. Wainer, and S. Rahimi, "ANNaBell Island: a 3D color hexagonal SOM for visual intrusion detection," *Internation Journal of Computer Science and Information Security*, vol. 9, no. 1, 2011, pp. 1–7.

[54] Z. Li, Y. Li, and L. Xu, "Anomaly intrusion detection method based on k-means clustering algorithm with particle swarm optimization," *2011 international conference of information technology, computer engineering and management sciences*. IEEE, 2011, vol. 2, pp. 157–161.

[55] P. Lichodzijewski, A. N. Zincir-Heywood, and M. I. Heywood, "Host-based intrusion detection using self-organizing maps," *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*. IEEE, 2002, vol. 2, pp. 1714–1719.

[56] L. Lindsay, S. Coleman, D. Kerr, B. Taylor, and A. Moorhead, "Explainable Artificial Intelligence for Falls Prediction," *International Conference on Advances in Computing and Data Sciences*. Springer, 2020, pp. 76–84.

[57] Z. C. Lipton, "The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery.," *Queue*, vol. 16, no. 3, 2018, pp. 31–57.

[58] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[59] Y. Liu, J. Sun, Q. Yao, S. Wang, K. Zheng, and Y. Liu, "A scalable heterogeneous parallel SOM based on MPI/CUDA," *Asian Conference on Machine Learning*. PMLR, 2018, pp. 264–279.

[60] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[61] Y. Ma, Z. Wang, H. Yang, and L. Yang, "Artificial intelligence applications in the development of autonomous vehicles: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, 2020, pp. 315–329.

[62] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable artificial intelligence (xai) to enhance trust management in intrusion detection systems using decision tree model," *Complexity*, vol. 2021, 2021.

[63] A. Marshan, "Artificial intelligence: Explainability, ethical issues and bias," *Annals of Robotics and Automation*, 08 2021, pp. 034–037.

[64] A. McDole, M. Abdelsalam, M. Gupta, and S. Mittal, "Analyzing CNN based behavioural malware detection techniques on cloud IaaS," *International Conference on Cloud Computing*. Springer, 2020, pp. 64–79.

[65] A. McDole, M. Gupta, M. Abdelsalam, S. Mittal, and M. Alazab, "Deep Learning Techniques for Behavioural Malware Analysis in Cloud IaaS," *Malware Analysis using Artificial Intelligence and Deep Learning*, Springer, 2021.

[66] L. Mohammadpour, T. C. Ling, C. S. Liew, and C. Y. Chong, "A convolutional neural network for network intrusion detection system," *Proceedings of the Asia-Pacific Advanced Network*, vol. 46, no. 0, 2018, pp. 50–55.

[67] J. D. Moore and W. R. Swartout, *Explanation in expert systemss: A survey*, Tech. Rep., UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST, 1988.

[68] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[69] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[70] Z. Muda, W. Yassin, M. Sulaiman, and N. Udzir, "Intrusion detection based on K-Means clustering and Naïve Bayes classification," *2011 7th international conference on information technology in Asia*. IEEE, 2011, pp. 1–6.

[71] S. Neupane, J. Ables, W. Anderson, S. Mittal, S. Rahimi, I. Banicescu, and M. Seale, "Explainable Intrusion Detection Systems (X-IDS): A Survey of Current Methods, Challenges, and Opportunities," *arXiv preprint arXiv:2207.06236*, 2022.

[72] S. Neupane, J. Ables, W. Anderson, S. Mittal, S. Rahimi, I. Banicescu, and M. Seale, "Explainable Intrusion Detection Systems (X-IDS): A Survey of Current Methods, Challenges, and Opportunities,", 2022.

[73] S. Neupane, I. A. Fernandez, W. Patterson, S. Mittal, and S. Rahimi, "A Temporal Anomaly Detection System for Vehicles utilizing Functional Working Groups and Sensor Channels," *IEEE International Conference on Collaboration and Internet Computing (IEEE CIC 2022)*, 2022.

[74] E. Oja and S. Kaski, *Kohonen maps*, Elsevier, 1999.

[75] J. Ong and S. M. R. Abidi, "Data Mining Using Self-Organizing Kohonen Maps: A Technique for Effective Data Clustering & Visualization," *IC-AI*, 1999.

[76] V. Pachghare, P. Kulkarni, and D. M. Nikam, "Intrusion detection system using self organizing maps," *2009 International Conference on Intelligent Agent & Multi-Agent Systems*. IEEE, 2009, pp. 1–5.

[77] E. J. Palomo, E. Domínguez, R. M. Luque, and J. Muñoz, "A new GHSOM model applied to network security," *International Conference on Artificial Neural Networks*. Springer, 2008, pp. 680–689.

[78] E. J. Palomo, E. Domínguez, R. M. Luque, and J. Munoz, "A self-organized multiagent system for intrusion detection," *International Workshop on Agents and Data Mining Interaction*. Springer, 2009, pp. 84–94.

[79] G. Pang, C. Ding, C. Shen, and A. v. d. Hengel, "Explainable Deep Few-shot Anomaly Detection with Deviation Networks," *arXiv preprint arXiv:2108.00462*, 2021.

[80] R. Panigrahi and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems," *International Journal of Engineering & Technology*, vol. 7, 3 2018, pp. 479–482.

[81] X. Qu, L. Yang, K. Guo, L. Ma, T. Feng, S. Ren, and M. Sun, "Statistics-enhanced direct batch growth self-organizing mapping for efficient DoS attack detection," *IEEE Access*, vol. 7, 2019, pp. 78434–78441.

[82] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, 2019, p. 9.

[83] Raytheon, "Cyber Security Operations Center (CSOC),", 2017.

[84] B. C. Rhodes, J. A. Mahaffey, and J. D. Cannady, "Multiple self-organizing maps for intrusion detection," *Proceedings of the 23rd national information systems security conference*. MD Press Baltimore, 2000, pp. 16–19.

[85] M. T. Ribeiro, S. Singh, and C. Guestrin, "" Why should i trust you?" Explaining the predictions of any classifier," *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[86] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive science*, vol. 9, no. 1, 1985, pp. 75–112.

[87] M. Salem and U. Buehler, "An enhanced GHSOM for IDS," *2013 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2013, pp. 1138–1143.

[88] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*, Springer Science & Business Media, 2011.

[89] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. C. Platt, "Support Vector Method for Novelty Detection," *NIPS*, 1999.

[90] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSp*, vol. 1, 2018, pp. 108–116.

[91] A. Sharma and S. K. Sahay, "Evolution and detection of polymorphic and metamorphic malwares: A survey," *arXiv preprint arXiv:1406.7061*, 2014.

[92] E. H. Shortliffe, *MYCIN: a rule-based computer program for advising physicians regarding antimicrobial therapy selection.*, Tech. Rep., Stanford Univ Calif Dept of Computer Science, 1974.

[93] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, "BAT: Deep learning methods on network intrusion detection using NSL-KDD dataset," *IEEE Access*, vol. 8, 2020, pp. 29575–29585.

[94] B. Subba, S. Biswas, and S. Karmakar, "Intrusion detection systems using linear discriminant analysis and logistic regression," *2015 Annual IEEE India Conference (INDICON)*. IEEE, 2015, pp. 1–6.

[95] M. Szczepański, M. Choraś, M. Pawlicki, and R. Kozik, "Achieving explainability of intrusion detection system by hybrid oracle-explainer approach," *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[96] H. M. Tahir, A. M. Said, N. H. Osman, N. H. Zakaria, P. N. M. Sabri, and N. Katuk, "Oving K-means clustering using discretization technique in network intrusion detection system," *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*. IEEE, 2016, pp. 248–252.

[97] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee, 2009, pp. 1–6.

[98] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," 2009, pp. 1–6.

[99] M. Vasighi and H. Amini, "A directed batch growing approach to enhance the topology preservation of self-organizing map," *Applied Soft Computing*, vol. 55, 2017, pp. 424–435.

[100] M. Wang, K. Zheng, Y. Yang, and X. Wang, "An explainable machine learning framework for intrusion detection systems," *IEEE Access*, vol. 8, 2020, pp. 73127–73141.

[101] C. S. Wickramasinghe, K. Amarasinghe, D. L. Marino, C. Rieger, and M. Manic, "Explainable unsupervised machine learning for cyber-physical systems," *IEEE Access*, vol. 9, 2021, pp. 131824–131843.

[102] C. Wu, A. Qian, X. Dong, and Y. Zhang, "Feature-oriented Design of Visual Analytics System for Interpretable Deep Learning based Intrusion Detection," *2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE, 2020, pp. 73–80.

[103] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied soft computing*, vol. 10, no. 1, 2010, pp. 1–35.

[104] C. Xu, J. Shen, X. Du, and F. Zhang, "An intrusion detection system using a deep neural network with gated recurrent units," *IEEE Access*, vol. 6, 2018, pp. 48697–48707.

[105] Y. Yang, D. Jiang, and M. Xia, "Using improved GHSOM for intrusion detection," *Journal of Information Assurance and Security*, vol. 5, 2010, pp. 232–239.

[106] L. Yuan, *Implementation of self-organizing maps with Python*, University of Rhode Island, 2018.

[107] M. E. Zarlenga, Z. Shams, and M. Jamnik, "Efficient decompositional rule extraction for deep neural networks," *arXiv preprint arXiv:2111.12628*, 2021.

[108] G. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, 2000, pp. 451–462.

[109] J. R. Zilke, E. Loza Mencía, and F. Janssen, "Deepred–rule extraction from deep neural networks," *Discovery Science: 19th International Conference, DS 2016, Bari, Italy, October 19–21, 2016, Proceedings 19*. Springer, 2016, pp. 457–473.