technologies

MDPI

Article

A Hierarchical Machine Learning Method for Detection and Visualization of Network Intrusions from Big Data

Jinrong Wu, Su Nguyen, Thimal Kempitiya and Damminda Alahakoon *

Research Centre for Data Analytics and Cognition, La Trobe University, Bundoora, VIC 3083, Australia; melody.wu@latrobe.edu.au (J.W.); su.nguyen@rmit.edu.au (S.N.); t.kempitiya@latrobe.edu.au (T.K.)
* Correspondence: d.alahakoon@latrobe.edu.au

Abstract: Machine learning is regarded as an effective approach in network intrusion detection, and has gained significant attention in recent studies. However, few intrusion detection methods have been successfully applied to detect anomalies in large-scale network traffic data, and low explainability of the complex algorithms has caused concerns about fairness and accountability. A further problem is that many intrusion detection systems need to work with distributed data sources in the cloud. In this paper, we propose an intrusion detection method based on distributed computing to learn the latent representations from large-scale network data with lower computation time while improving the intrusion detection accuracy. Our proposed classifier, based on a novel hierarchical algorithm combining adaptability and visualization ability from a self-structured unsupervised learning algorithm and achieving explainability from self-explainable supervised algorithms, is able to enhance the understanding of the model and data. The experimental results show that our proposed method is effective, efficient, and scalable in capturing the network traffic patterns and detecting detailed network intrusion information such as type of attack with high detection performance, and is an ideal method to be applied in cloud-computing environments.

Keywords: clustering; distributed artificial intelligence; cloud computing; network problems; network-level security and protection; parallel processing

Citation: Wu, J.; Nguyen, S.; Kempitiya, T.; Alahakoon, D. A Hierarchical Machine Learning Method for Detection and Visualization of Network Intrusions from Big Data. *Technologies* **2024**, *12*, 204. https://doi.org/10.3390/technologies12100204

Academic Editor: Mohammed Mahmoud

Received: 15 July 2024
Revised: 24 September 2024
Accepted: 9 October 2024
Published: 17 October 2024

Correction Statement: This article has been republished with a minor change. The change does not affect the scientific content of the article and further details are available within the backmatter of the website version of this article.

1. Introduction

The dramatic increase in Internet use has brought many security concerns about network data integrity, availability, and confidentiality. Nowadays, malicious users continuously observe and analyze the communication networks for possible vulnerabilities to gain access to the system without authorization. As such, detecting anomalies in communication networks, especially cybersecurity intrusion, is a major challenge for researchers [1]. Intrusion detection consists of monitoring and analyzing network traffic patterns and log information, and scrutinizing for suspicious patterns. It allows accurate intrusion identification and reporting to the proper authorities for the enhancement of network security. Since many organizations depend on cloud computing due to its distributed nature and scalability, it is highly advantageous for intrusion detection systems to be distributed and scalable.

There are four main types of network intrusion attacks, namely Denial of Services (DoS), Remote to Local (R2L), User to Root (U2R), and Probing. DoS is one of the most frequent types of attack where attackers make network resources unavailable to legitimate users. There are many types of DoS attacks, such as SMURF and NEPTUNE, and they can be distinguished from each other according to their behavior on network resource consumption [2,3]. R2L normally happens when attackers send a set of packets to another machine through a network and attempt to gain full access illegally as a local user after exploiting the vulnerability of the network, while U2R takes place when the attackers try to access the network resources as normal users and after several attempts become full-access

users [2]. A probing attack refers to attackers attempting to gather information about the network and find vulnerabilities for further illegal use [2].

Based on where these intrusions take place, intrusion detection can be classified into two categories. Host-based intrusion detections focus on monitoring activities with a particular host, whereas network-based intrusion detections look for intrusions from network traffic. In general, two main types of approaches are applied in intrusion detection: misuse detection and anomaly detection. Misuse detection attempts to build models for abnormal behaviors with known attacks and thus is vulnerable to novel attacks, while anomaly detection targets differentiate normal and abnormal behaviors, results in a sacrifice in the false positive rate since it may not fully reflect the dynamic nature of network systems [4]. As such, an ideal intrusion detection technique should be more robust to the changing network environment while maintaining a minimum possible false positive rate [4,5].

Many machine learning (ML) methods have been proposed to detect intrusions over networks [2], and many sophisticated ML algorithms such as artificial neural networks have gained significant attention among the others since they have shown high detection capability in intrusion detection tasks [6,7]. However, there is a need for more extensive research on machine learning techniques in the unsupervised learning paradigm which can cater to unforeseen attacks and also cater to the needs of distributed and scalability requirements [8]. A further key requirement is the complex nature of most current models which makes them hard or even impossible to interpret by decision makers and thus brings concerns about their fairness, accountability, and transparency, especially in real-life applications [9]. As a result, network operators may end up with low trust in the decisions made by the ML algorithms even if they have shown high accuracy. Furthermore, most of the existing studies focus on the detection of high-level intrusion types, but the detection of more detailed information can be more useful to users; for example, whether the attack is a DoS or a R2L attack, and whether the attack is a SMURF attack or a NEPTUNE attack in the DoS attack domain, can help the user instantly locate and fix the problem. However, detecting attack types at a more detailed level is a more challenging task and can result in the reduction of model accuracy, due to the increasing complexity of relations among attacks and the higher possibility of imbalanced data distribution for different attacks [10–12]. Last, many traditional anomaly detection methods, such as K-Nearest Neighbor and Self-Organizing Maps, failed to work well on large-volume networks because of the high computation cost [8,10,11], so a scalable solution such as distributed computing or its specialized form such as cloud computing is needed.

Thus, the aim of this paper is to develop a novel intrusion detection method to accurately identify intrusion attack types in detail and improve the detection rate of minority classes, while maintaining a low cost on computation time, catering to distributed computing environments, and explaining the data patterns and decisions made by the model, for large volumes of network data.

The rest of the paper is organized as follows. Related work is presented in Section 2, and Section 3 explains our proposed methods in detail. Section 4 introduces the detailed implementation settings and results when applying the proposed method to two network traffic datasets. Finally, in Section 5, we draw conclusions from the reported research.

2. Background

2.1. Previous Resources

Many different types of algorithms have already been applied to solve anomaly detection problems in intrusion detection [13].

Statistical methods include the statistical changepoint detector, which is one of the commonly applied algorithms that has proven to be very efficient in network anomaly detection [14,15]. Recently, with the increasing use of machine learning algorithms, several surveys [16,17] categorized the intrusion anomaly detection methods and provided a detailed description of all these methods, including statistical and machine learning

methods. These surveys revealed that detection implementation through artificial neural networks (ANNs) [18] received more attention due to a comparatively better performance in detection tasks. Furthermore, several studies have tested the effectiveness of different machine learning algorithms with two popular intrusion detection datasets, the KDD Cup 1999 dataset [19–22] and its improved version NSL-KDD dataset [21–23]. In particular, Laskov et al. [19] applied both supervised and unsupervised methods, such as K-Nearest Neighbor, Multi-Layer Perceptron, Support Vector Machine (SVM), K-Means Clustering, and Single Linkage Clustering, and showed that unsupervised and semi-supervised learning approaches are not affected by unknown attacks while maintaining similar performance in intrusion detection with supervised learning approaches. Several semi-supervised approaches were introduced in intrusion detection [24,25], and a graph-based semi-supervised method [26] showed that it could achieve better performance than the traditional supervised learning methods.

Many unsupervised learning techniques have been applied and proven to be effective in intrusion detection. Vinutha and Poornima [27] applied a simple K-Means Clustering method and assigned Manhattan distance metrics to reduce the number of incorrectly classified instances, leading to an 82.27% detection rate on the NSL-KDD dataset. Li et al. [28] proposed a K-Means Clustering method with an improved Euclidean distance formula facilitating automatic calculation of the distance between the data and cluster center, and thus could adjust the number of clusters when the distance exceeded the threshold. The method showed a detection rate of 90% when applied to the KDD Cup 1999 dataset to identify four main attacks (DoS, Probe, R2L, and U2R). Shyu et al. [29] and Kwitt and Hofmann [30] presented an anomaly detection technique where the authors performed principal component analysis (PCA) to estimate the principal components from the covariance matrix.

Self-Organization Maps (SOMs) is another typical unsupervised algorithm based on neural networks, and its extended methods were also proven to be effective in network anomaly detection [31]. Qu et al. [10] and Choksi et al. [11] reviewed the existing studies of SOM-based methods for network intrusion, and demonstrated the effectiveness of SOM on different network intrusion datasets, while Ramadas et al. [32] applied a SOM-based method to successfully detect anomalies in a self-collected network traffic dataset. In addition, the Kohonen Self-Organization Map (K-Map) algorithm [33] and a hierarchical SOM-based intrusion detection algorithm [34] were validated on the KDD Cup 1999 dataset with a detection rate of between 90.4% and 93.46% in the testing dataset. Tan et al. [35] improved SOM with k-means algorithms and this method achieved a 94% binary classification accuracy when tested on the NSL-KDD dataset. Adaptive Resonance Theory, a neural-network-based unsupervised algorithm which is very similar to SOM, achieved a 97% precision in an intrusion detection setting with 27 anomaly types based on around 5000 training packets and 3000 testing packets [36].

However, while SOM-based intrusion detection methods showed high performance, reviews of SOM-based methods [10,11] pointed out some existing challenges: (1) although detection performance can be good for binary classification (normal vs. attack), the detection accuracy for identifying the detailed attack type is insufficiently accurate and the false positive rate is relatively high; (2) existing SOM-based intrusion detection algorithms are time-consuming.

2.2. Big Data Processing in Network Anomaly Detection

Big data issues can significantly impact network data. For example, 1-Gbps-sustained network traffic may lead to a big data challenge for intrusion detection when applying deep packet inspection [37]. Several distributed computing-based platforms have been applied to address big data problems, including Apache Spark [38], and Hadoop and Storm [39]. Among them, Apache Spark, with a multi-staged in-memory processing scheme, is considered the best choice among researchers as the data processing speed of Apache Spark is 100 times faster than map-reduce-based processing tools, such as Hadoop [39]. Apache Spark can also run in the cloud and it provides support for multiple

programming languages, such as Java and Python, with a user-friendly API and shell in Python and has proven to be very effective in many big data analysis tasks [39,40].

3. The Proposed Method

Network traffic data normally have very high dimensions, which poses significant challenges for ML algorithms both in terms of intrusion accuracy and interpretability [40]. To understand the high dimensional network traffic data in intrusion detection and avoid the effects of the curse of dimensionality [41], dimension reduction is the most popular approach [42]. Traditional dimension reduction techniques such as PCA or linear discrimination analysis (LDA) are not very effective in this case because of the complexity of non-linear relations in features. Meanwhile, advanced techniques such as auto-encoders or deep auto-encoders have trouble learning effective latent representation due to the mixing of continuous and categorical data in network data, while SOM-based algorithms are known to be time-consuming and insufficiently accurate in detecting anomalies with unbalanced network data.

To overcome the above challenges, we extend the Distributed Growing Self-Organizing Map (DGSOM) algorithm proposed by Jayaratne et al. [40] to implement two classification approaches: a max-count-based classification algorithm and a hierarchical classification algorithm, for network intrusion anomaly detection. To address the issue of unbalanced network data in SOM algorithms, we employ additional classification algorithms specifically for the minority classes. The overall process of the proposed approach is illustrated in Figure 1. In Figure 1, either the max count or hierarchical classification approach will be selected for intrusion detection.

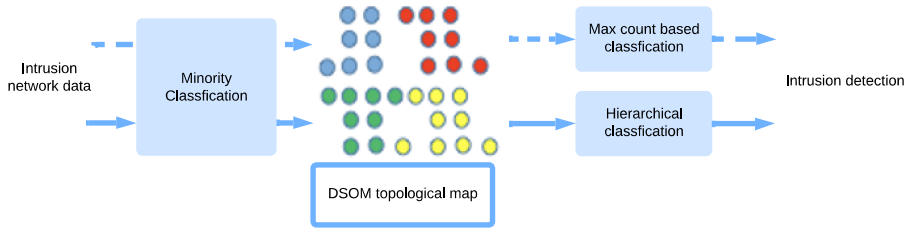


Figure 1. The overall framework of the proposed detection and visualization framework.

Through our proposed method, each individual input is embedded into a topographic map using the Growing Self-Organizing Map (GSOM) [37] algorithm and thus each individual can have a two-dimensional latent representation based on this topological map. Since GSOM is a model-free algorithm and does not rely on any strict assumptions, it is a powerful algorithm for learning latent representations of arbitrary datasets. DGSOM is an extension of GSOM to cope with high dimensional big datasets based on distributed computing, thus it is ideal for distributed cloud computing environments. Through the DGSOM algorithm, several topological maps produced by GSOM are merged together with the help of the parallel processing tool of Apache Spark by Sammon's projection, and in the merged map, a close neighbor of a data point represents a close relationship between them. Since the latent representations (or self-organized maps) are obtained by DGSOM in an unsupervised manner, it is less biased towards certain types of attacks and improves the generalization ability of ML algorithms. Furthermore, the obtained map is an effective way to visualize network patterns.

3.1. Self-Organization-Based Networking Data Processing

GSOM, a dynamic variant of the Self-Organizing Map (SOM) algorithm [43], is the basis of our proposed network anomaly detection method. With its self-structuring ability, GSOM can address the major issue of SOM which is determining a suitable size and shape for the map [43]. Hence, it can identify the ideal map structure automatically without the user having to specify the same arbitrarily and has been demonstrated to be faster than the standard SOM algorithm [44]. Like the SOM, the GSOM also consists of a two-

dimensional lattice of neurons with a unique weight vector for each neuron. The GSOM learning algorithms contain two sequential processes, growing and smoothing. During the growing process, we initialize the map with a 2×2 lattice, with a random weight vector assigned to the four neurons. For a dataset with N input vectors, each input vector $x_i \in R^D$, where D is the dimensionality of the input, is added to the map and the neuron which is closest to the input by calculating the weight distance in (1) is denoted as the best matching unit (BMU).

$$E(t) = \sum_{i=1}^{N} ||x_i - w_{BMU}(x_{i_t})|| \quad (1)$$

$E(t)$ is the quantization error of iteration t. Based on this formula, new neurons are added to the map before $E(t)$ exceeds threshold GT, which is calculated based on the predefined spread factor (SF, which controls the spread of the GSOM), the number of dimensions D, and the size of the dataset N, in (2).

$$GT = -D \times \ln(N) \times \ln(SF) \quad (2)$$

After each iteration, the weight vector of a BMU and its neighbors will be updated by (3)

$$w_k(t+1) = \sum_{i=1}^{N} (w_k(t) + \alpha h_{ck}(t)(x_i - w_k(t))) \quad (3)$$

w_k is the weight vector of the neuron k, α is the learning rate, and $h_{ck}(t)$ is the neighborhood function (in our study it is set as Gaussian) for preserving the topological relationships of the neurons within the map.

During the smoothing process, the weight of each vector is further fine-tuned but without any new neurons being added to the existing map.

3.2. Distributed-GSOM-Based Parallel Processing for Big Data

Same as SOM [10,32], the major limitation within the current GSOM algorithm is the time complexity resulting from the algorithm framework. This has restricted the application of GSOM in some datasets that are very large in size or with high dimensions. To address this issue, many existing methods have been proposed with two main directions: (1) GPU-based network parallelized models and (2) batch SOM-algorithm-based data parallelized models. However, the former methods have been found to be restricted by maximum processors equal to the number of SOM nodes, resulting in limiting the processing improvement as well as underutilization of GPU capacity, while the latter ones require batch SOM where weight synchronization after each iteration becomes a major bottleneck on speed and results in the reduction of final map quality [44,45]. Thus, to address these existing issues, we applied the DGSOM algorithm [40], an improved GSOM based on Apache Spark which does not have to synchronize after each iteration and preserves the quality of the final map with the use of the online variant. The DGSOM algorithm has demonstrated super-linear speedup while generating topographic maps on underlying data to enable a speedup of 200 when compared to SOM; at the same time, it could preserve the topology of the data even better than the original GSOM algorithm with the use of Sammon's projection [44].

The input data are first randomly partitioned into small datasets and each dataset was trained by the GSOM algorithm with parallel processing, as shown in Figure 2. Since each trained GSOM cannot communicate during the training process, we can usually find redundant neurons, which represent similar patterns in different GSOMs after training. As such, we utilize a redundancy reduction method to identify and remove these redundant neurons. Here, Sammon's projection [46] is applied to assist in merging each single-trained GSOM. Sammon's projection is a method that is usually used to map the high dimensional data into a lower-dimensional space while maintaining the topological relationships within the input data. This merging process has been conducted several times to minimize the error in map combination.

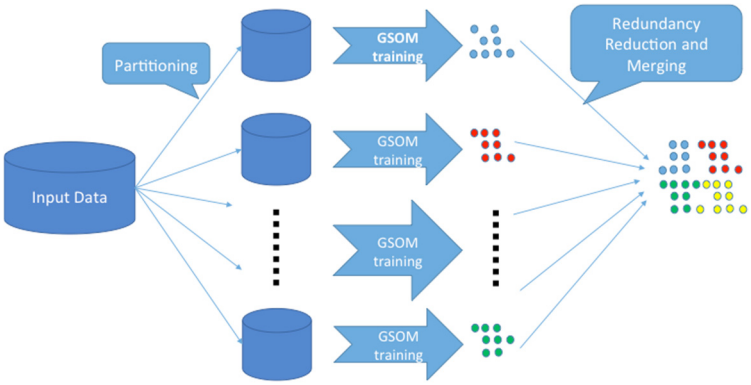


Figure 2. The overall framework of the GSOM training processes with parallel programming.

We use five main operations provided by Apache Spark to implement this method: (1) a map operation labels each input with a random integer from 1 to *k*, where *k* represents the number of partitioned datasets and k's value is a hyperparameter for the DGSOM algorithm; (2) a *reduceByKey* operation collects the data inputs sharing the same label and form as the partitioned dataset; (3) after (2), another map operation is applied to run the GSOM algorithm on each partitioned dataset simultaneously; (4) a TreeReduce operation then removes the redundant neurons iteratively from all the maps; and (5) finally, a list of neurons pertaining to all the non-redundant neurons from multiple maps are merged with Sammon's projections to obtain the final map.

3.3. Hierarchical and Minority Classifiers Training

After a GSOM is trained with the DGSOM algorithm, we are able to map the training inputs to the map, so that each training input has a matched GSOM neuron (BMU), and we can obtain a GSOM together with its matched training inputs information (training GSOM).

There are three different types of neurons in the training GSOM, namely, pure neurons, not-pure neurons, and empty neurons (due to the growing mechanism of GSOM, training GSOMs may contain empty neurons). In this paper, the neurons with all matched training inputs of only one class are referred to as pure neurons, and the neurons with no matched training input are defined as empty neurons. The rest of the neurons, containing the matched training inputs of more than one class, are considered not-pure (impure) neurons. This means, if a node is a pure node, it can reflect that the GSOM algorithm correctly clusters the data samples mapped to this node, and as such more confidence can be given to the clustering decision for this node. As a result, the class label that is shared by all matched training data inputs in this node can be adopted as the predicted class label for all the testing data samples that are mapped to this node. Vice versa, if a node is impure, less confidence can be given to the GSOM algorithm clustering decision, and some further classification steps will be required to help correctly classify the test data inputs when they are mapped to the node.

In this paper, we propose two different approaches to improve classification performance for these impure neurons; one is a max-count-based classification approach and another is a hierarchical approach. Since the max-count-based classification approach does not need the support of any other machine learning classifiers, no extra training is needed for the max count approach.

For the hierarchical approach, we train several classifiers based on the matched training inputs in not-pure neurons with traditional self-explainable classification methods such as logistic regression or decision tree models. The detailed training procedures to build the classifiers are shown in Algorithm 1. Neurons that contain the same label set are grouped in the same neuron group. For example, we assume there is a neuron containing the training inputs with a class label set of {1, 2, 4}. If neurons share the same class label set of {1, 2, 4} or the label set of this neuron is a subset of {1, 2, 4}, such as, {2, 4}, {1, 2}, or {1, 4}, these neurons will be grouped in the same neuron group. After this process, training data in the

same neuron group is gathered to train a separate classifier for this neuron group. After these processes, all not-pure neurons are assigned with a classifier and are flagged with the name of this specific classifier. In the end, all the classifiers are stored locally for the convenience of further classification.

Algorithm 1: Training Classifiers for Not-Pure Neurons for the Hierarchical Approach

Inputs: Matched training inputs for not-pure neurons

Return: Classifiers for not-pure neurons

```
Group neurons containing the same classes
for each neuron group:
    Classifier training data ← fetch all the training inputs belonging to the neurons in
                               this group
    Build a classifier for the training data
    Save the classifier
```

In addition, to deal with the very unbalanced data issue commonly identified in network data, we apply additional classifiers for the minority classes to help improve the detection of the minority classes. Since the additional classifier for the minority class is trained with the help of the self-explainable machine learning methods applied in the hierarchical approach (decision tree or logistic regression algorithm), this minority class handling process is only implemented for the hierarchical approach. As shown in Algorithm 2, if the percentage of data samples for a class in the total training data samples is below the threshold, we will train a classifier to help identify this minority class in advance. We resampled the dataset and trained a classifier for each of these minority classes with 10-fold cross-validation. For example, if class A is identified as a minority class with 100 data samples in the training set, we will randomly select another 100 samples from the rest of the datasets and train a linear model or decision tree model based on the combined dataset.

Algorithm 2: Training Classifiers to Identify Minority Class

Inputs: Matched training inputs for not-pure neurons; Threshold of dealing with minority label class

Return: Classifiers for not-pure neurons

```
for each label in classification labels:
if percentage of data samples in a class < threshold:
    Resample the dataset and train a model to classify this class and the rest of the
    other classes
```

3.4. GSOM-Based Anomaly Detection

To detect anomalies in the testing set, testing data are mapped to the GSOM obtained from Section 3.2, and each testing input has its matched GSOM neuron (BMU), which can also be identified in the training GSOM. The overall procedures of the GSOM- based classification algorithm are shown in Algorithm 3 and Figure 2. In Figure 3, each step in the detection intrusion process is identified and in step 3 the max count or hierarchical approach is used which is identified in Algorithms 4 or 5. Details of steps to implement the classification task for each testing input are as follows:

Step 1. If it is a hierarchical approach and we decide to deal with minority class, a trained classifier will be applied to identify whether it is predicted as a minority class. If yes, we will assign the predicted minority class label to it, and the prediction for this testing input is completed. If it is not predicted as any of the minority classes, it will continue to be processed in the next step.

Step 2. If the matched neuron for this testing input is a pure neuron in the training GSOM, this testing input is considered to share the same class label as the training data

mapped to the same neuron (since they share the same latent space representations in the GSOM space mapping). Otherwise, it will continue to be processed in the next step.

Step 3. If the matched neuron for this testing input is a not-pure neuron in the training GSOM, this testing input will be classified further with the approach defined in Algorithm 4 or 5 (reasons for this can be found in Section 3.3). Otherwise, it will continue to be processed in the next step.

Step 4. If all the above requirements are not met for a testing input, it implies that the matched neuron for this testing input is an empty neuron in the training GSOM with no matched training data; as such, it will be mapped to the closest neuron which is not empty, and depending on the condition of the closest neuron, testing input is further classified.

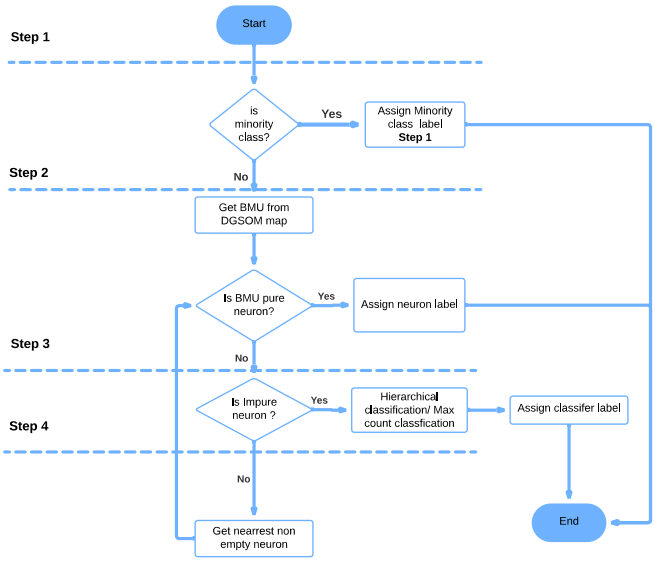


Figure 3. Individual testing data point process flow chart.

Algorithm 3: GSOM-based Classification

Inputs: Training inputs data; Test inputs data; Neurons in the training GSOM; Test inputs matched neurons; Flag of whether dealing with minority label class

Return: Predicted classification label for each testing input

for each neuron in the test inputs matched neurons:
  for each testing input in the neuron:
    if Flag of whether dealing with minority label class is True **and** testing input is predicted as a minority class label by minority classifiers defined in Algorithm 2 (only in hierarchical approach):
      update predicted classification label as the predicted label by the minority classifiers
    elif the neuron in training GSOM map is pure:
      update predicted classification label as the neuron representative class label
    elif the neuron is not pure:
      update predicted classification label for this test input based on a classification approach (Algorithms 4 and 5)
    else (the neuron is empty):
      search for the closest neuron (based on the latent space distance) which is not empty in the training map and update the predicted classification label for this test input based on a classification approach

3.4.1. Max-Count-Based Approach

One way to classify the not-pure neurons is simply by using a max count method, where the most frequent class in the training inputs of this neuron is defined as the class label in testing data. The detailed procedure is shown in Algorithm 4.

Algorithm 4: Max Count Approach
Inputs: A test input X_i; Neurons in the training GSOM; Training inputs data
Return: Predicted class labels with a max frequency count for the test input, P_{x_i}
neuron k $\leftarrow$ the matched neuron of test input X_i in training GSOM map class label $\leftarrow$ class label with max frequency count among the matched training inputs of the neuron k $P_{x_i} \leftarrow$ class label

3.4.2. Hierarchical Approach

Another approach is to apply a traditional explainable classification method such as logistic regression and decision tree models to classify testing data on the not-pure neurons.

The overall implementation process of the hierarchical approach is shown in Algorithm 5. When a test input is presented and mapped to a neuron with a classifier flag, we will use the specific classifier assigned to that neuron to classify this testing input.

Algorithm 5: Hierarchical Approach
Inputs: A test input X_i; Neurons in the training GSOM; pre-trained classifiers
Return: Predicted class labels based on the classification result from classifier, P_{x_i}
neuron $k \leftarrow$ the matched neuron of test input X_i in training GSOM map classifier $Cl_k \leftarrow$ the classifier that neuron k was assigned to among all pre-trained classifiers $P_{x_i} \leftarrow$ predicted class for X_i based on classifier Cl_k

3.5. Explainability Through Visualization

The GSOM algorithm uses self-organizing and self-structuring to learn the structure of the data as a topographic map, leading to the construction of a two-dimensional latent representation of the data. A key advantage of this representation is that relationships that exist across multiple dimensions (variables), and are not comprehensible to humans, are captured in a two-dimensional (or even 3) map which preserves the topological relationships. This latent representation and topographic map can be visualized in two dimensions. Visualization is an important approach used by humans as a tool for explanation [47,48], and both SOM and GSOM algorithms are identified as inherently explainable algorithms [49,50]. The GSOM, with its ability to grow the structure dynamically, has an advantage over the SOM for capturing and representing input patterns. The proposed intrusion detection algorithm utilizes the GSOM topographic map as the knowledge base to determine intrusion detection. In this work, we consider the global interpretability of unsupervised algorithms [51]. Global interpretability is concerned with explaining why a set of data points belongs to a specific cluster, and in our work, we focus on explaining why a set of data points belongs to a specific neuron through the 2D topographic map visualization. The GSOM not only highlights the groupings within the data by representing similar input mapped to neighboring nodes, it also presents the overall similarities and differences in the input as neighborhood relationships within the map. Therefore, the GSOM can be considered as an unsupervised machine-learning-generated visualization of the otherwise hidden relationships and patterns in a dataset.

4. Experiment and Results

This section presents the datasets used for the experiment and the experiment results with its performance metrics and detailed results analysis.

4.1. The Dataset

In order to prove the effectiveness of our proposed method, we have set up two intrusion anomaly detection experiments based on the most commonly applied datasets, the KDD CUP 99 dataset [20,52] and its improved version the NSL-KDD dataset [23]. We

applied the original KDD CUP 99 dataset to detect a detailed level network-attack-type detection (with 23 classes) and applied the NSL-KDD dataset to a 5-class attack-type detection (DoS, Probe, R2L, U2R, and normal). Since the original KDD CUP 99 dataset was found to have some issues which may not reflect the real performance of a detection model [23], we used detection performance based on the NSL-KDD dataset to compare with the other state-of-the-art detection models.

4.1.1. KDD CUP 99 Dataset

The KDD CUP 99 dataset is built based on the data captured in the DARPA'98 IDS evaluation program [53]. The KDD CUP 99 training dataset contains 4,898,431 input vectors with 41 features for each vector and the testing dataset consists of 311,027 input vectors. The KDD'99 training dataset is a fully labeled dataset with 22 types of attack apart from the normal state type; thus, a total of 23 different labels and the testing dataset contains a total of 38 labels.

In the experiment, we aimed to classify the dataset based on its very detailed attack types rather than binary categories (normal or attack) or the five major categories, normal, DoS, R2L, U2R, and Probing, without dealing with minority classes, so we removed the attack types and their related testing data that were never shown in the training dataset. A detail demographic distribution of the training and testing data applied in our experiment are shown in Table 1. In terms of data preprocessing, we transformed categorical features into dummy variables and applied Min–Max normalization on the training input to normalize the numerical features in the training dataset. The same methods were applied on the test dataset; thus, all vectors of the training and testing data ranged between 0 and 1.

Table 1. Demographic distribution of the Kdd'99 training and testing dataset in our experiment setting.

Types	Training Dataset	Testing Dataset
smurf.	2,807,886	164,091
neptune.	1,072,017	58,001
normal.	972,781	60,591
satan.	15,892	1633
ipsweep.	12,481	306
portsweep.	10,413	354
Nmap.	2316	84
back.	2203	1098
Wareclient.	1020	0
teardrop.	979	12
pod.	264	87
guess_passwd.	53	4367
buffer_overflow.	30	22
land.	21	9
warezmaster.	20	1602
imap.	12	1
rootkit.	10	13
loadmodule.	9	2
ftp_write.	8	3
multihop.	7	18
phf.	4	2
perl.	3	2
spy	2	0

4.1.2. NSL-KDD Dataset

Statistical analysis of the KDD CUP 99 dataset showed that there were some important issues which could affect the performance of the systems [23]. As such, to solve these issues, the NSL-KDD dataset is proposed by selecting records from the original KDD dataset [54].

It has three main advantages: (1) it has no redundant records in the training set, so the trained classifier can be less biased; (2) it has no duplicated record in the test set, so the detection rate is less biased; and (3) the number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD dataset, so the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of different learning techniques. The NSL-KDD training dataset contains around 125,973 input vectors with 41 features for each vector and the testing dataset consists of around 22,544 input vectors.

Same as the experiment based on KDD CUP 99, we aimed to detect detailed attack types in network intrusion, but since the first experiment was at a very detailed attack level and hard to compare with existing methods focusing on binary or five-class anomaly detection, we applied our method to detect five major categories in network traffic data, normal, DoS, R2L, U2R and Probing, which could still present a detailed attack type. To prepare the dataset for training and testing purposes, we transformed the categorical variables with dummy coding and applied Min–Max normalization to normalize both the training and testing data.

4.2. Evaluation of Detection Performance

In terms of evaluating the intrusion detection performance of our proposed method, we applied three well-accepted evaluation metrics for intrusion detection systems for each network event type, namely, detection rate, false positive rate, and accuracy [47,48], and used an overall accuracy to evaluate the overall performance of our method. The overall accuracy of the intrusion detection system is the ratio of the sum of normal and abnormal records correctly detected to the total size of the testing data. Detection rate, also referred to as the true positive rate, recall, or sensitivity, is a ratio of the number of correctly detected records in a class over the total number of records in that class. The false positive rate, also referred to as the false alarm rate (FAR), is the number of incorrectly detected alarms for a class divided by the total number of normal records in this class. The accuracy for an attack type is defined as the ratio of correctly classified samples for the attack type to the total number of data samples. All other evaluation results, true negative rate (specificity), false negative rate, and precision can be found in the Appendix. In order to investigate the scalability of our preprocessed method, we also record the time that is used to train the models.

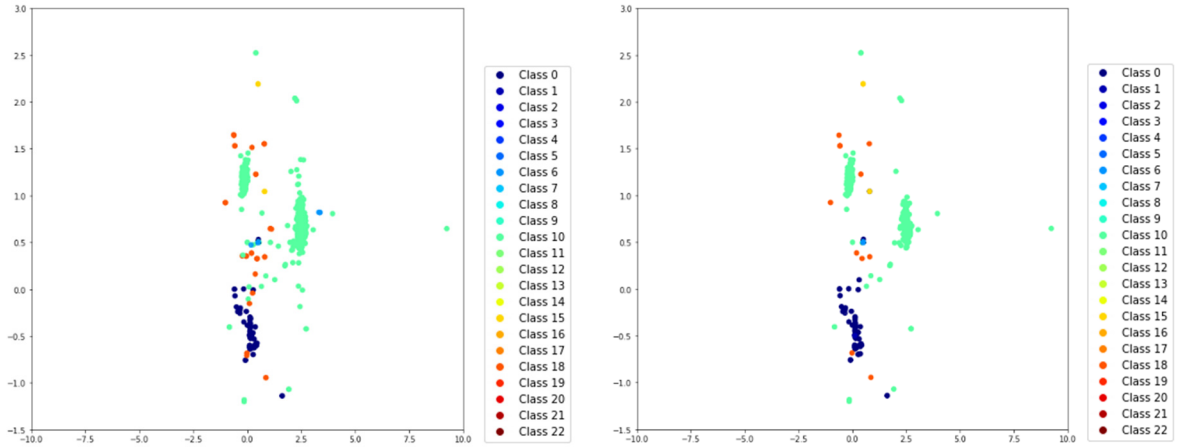
4.3. Configuration Settings

In our experiments, we have partitioned the dataset into 32 subsets for distributed processing, since the computing cluster contained 32 virtual computing cores. The number of iterations was set to 100 when training a GSOM in both growing and smoothing phases. The learning rate was 0.3 and the spread factor was 0.7. We trained the logistic regression and decision tree models with grid search and 10-fold cross-validation.

4.4. Results and Analysis

4.4.1. KDD CUP 99 Dataset

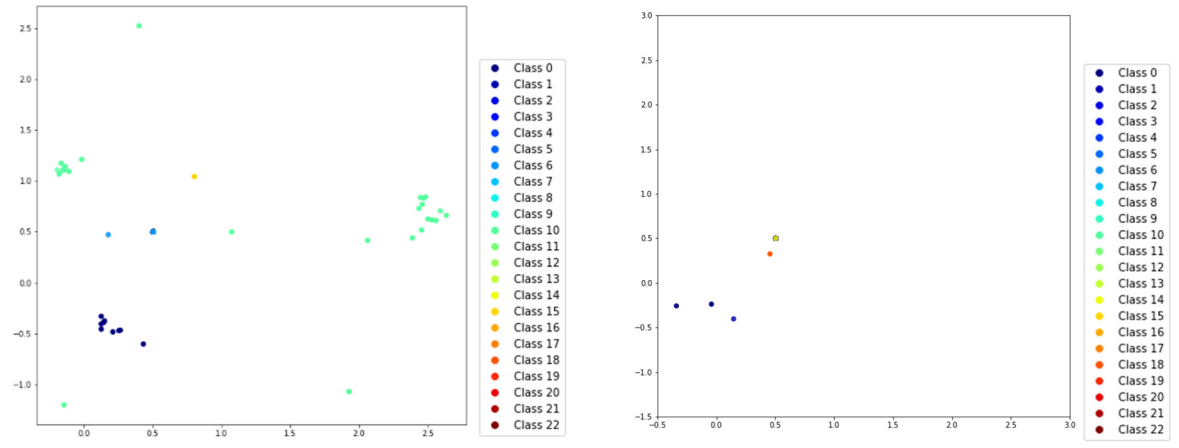
After training our detection model with the DGSOM algorithm, we generated a latent space representation map for both training and testing data of the KDD CUP 99 dataset, as shown in Figure 4. It can be identified that there is a significant clustering pattern in type normal (class 0) and type NEPTUNE (class 10). In addition, the training GSOM and testing GSOM have shown very similar clustering results. Most of the neurons show the same color at the same position in the training and testing map, indicating that DGSOM, together with the max count approach proposed in this paper, might be able to accurately classify the testing data, since the representative color of a neuron in the map is also generated following a max count technique.



(**a**) (**b**)

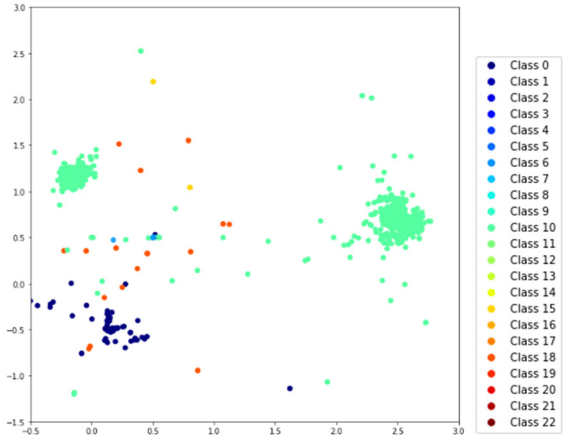
Figure 4. GSOM representation of the KDD CUP 99 training data (**a**) and testing data (**b**); Class 0 to Class 22 represent that type of normal, back, buffer_overflow, ftp_write, guess_passwd, imap, ipsweep., land, loadmodule, multihop, neptune, nmap, perl, phf, pod, portsweep, rootkit, satan, smurf, spy, teardrop, warezclient, and warezmaster, accordingly. The representative color for a neuron in the map is the class type which has the highest frequency count.

We also identified that there were a total of 12,607 neurons in the training GSOM, among which, 7917 were pure neurons (each neuron only contains data vectors of one class label), and 700 neurons were not-pure neurons. The remaining 3989 neurons were empty neurons. In the testing GSOM, we identified 4750 pure neurons, 140 not-pure neurons, and 7717 empty neurons. Most of the pure neurons in training and testing maps shared the same labels (as shown in Figure 4, since most of the neurons shown in the map were pure neurons), thus we were more interested to find some patterns within the not-pure neurons in both the training and testing map. As shown in Figure 5, we found that most of the not-pure neurons in the testing map were empty neurons in the training map, indicating a high possibility of misclassification in testing if data were mapped to these not-pure neurons. We further investigated the training GSOM for all neurons and focused on the area where we located the not-pure neurons in the testing map and proved our hypothesis that for not-pure neurons in the testing dataset, it would be hard to find any neuron neighbor in the training GSOM to help to conduct classification. Figure 5a,b only display the pure neurons, which results in some classes being missed and emphasizes the need for the proposed impure labeling process.



(**a**) (**b**)

Figure 5. *Cont.*



(**c**)

Figure 5. The magnified GSOM for not-pure neurons in training (**a**), the testing dataset, (**b**) and the training GSOM for all neurons focusing on the not-pure neuron areas (**c**).

When investigating the characteristics of these not-pure neurons, we found that the majority of these neurons contained more than four attack-type labels. Although these neurons contained only around 0.5% of the total training dataset, many input data vectors with low label frequency count in the training dataset were in these not-pure neurons and, it would be hard to further classify them, since capturing the patterns of such low frequency data is difficult.

Table 2 shows the classification results for different anomaly types. All three approaches showed a very high detection accuracy and low false positive rate, which mitigated the high false positive rate issue in existing SOM-based methods [10,11]. However, when it comes to detection rate, three of the approaches did not seem to be good enough for anomaly training types with less than 100 training cases. The hierarchical approach with decision trees performed better in terms of detection rate and this finding suggests the potential to use the hierarchical approach to improve the detection rate in our Experiment 2. Other evaluation results including true negative rate, false negative rate, and precision can be found in Table A1 in Appendix A.

Regarding the overall performance of the three proposed intrusion detection approaches, the max count approach achieved an accuracy of 97.33% in the intrusion detection experiment as shown in Table 3. The hierarchical classifiers of logistic regression and decision tree did not perform as we expected, and this might result from the multiple classification problems in the subset of data mapped to not-pure neurons that there were not enough cases to present the patterns for accurate classification. A further fine-tuning of the parameters or utilizing an unbalanced data-handling algorithm, such as the SMOTE, may help improve the performance of these hierarchical approaches.

In terms of computation time, the parallel implementation of DGSOM with Apache Spark reduced the execution time to less than 373 min across different approaches applied in our study (364.3 min for the max count approach, 367.36 min for the hierarchical approach with decision tree, and 369.8 min for the hierarchical approach with logistic regression), which significantly decreased the training time when compared with other neural-network-based methods [55]. Moreover, our proposed method achieved a better performance than the other detailed level classification methods applied on KDD'99 in both type-specific accuracy and false positive rate and achieved a better overall accuracy than the existing approaches, especially other SOM-based algorithms [2,31]. In addition, with the visualization and self-explanation ability from decision trees or logistic regression models, we can better understand the input data. It also should be noted that our method achieved very promising performance without utilizing any sampling techniques to cope with the imbalanced classes in the large-scale network dataset. The above results showed that

the latent representations obtained by DGSOM could effectively capture the topological relations of input data in an unsupervised manner and made this proposed method less vulnerable to imbalanced class issues in classification tasks.

Table 2. Detection rate and false positive rate evaluation for three approaches on the KDD CUP 99 dataset.

	Max Count Approach			Hierarchical Approach with Logistic Regression			Hierarchical Approach with Decision Tree		
Types	**Accuracy**	**False Positive Rate**	**Detection Rate**	**Accuracy**	**False Positive Rate**	**Detection Rate**	**Accuracy**	**False Positive Rate**	**Detection Rate**
smurf	0.9998	0.0003	0.9999	0.9998	0.0002	0.9999	0.9996	0.0008	0.9999
neptune	0.9876	0.0136	0.9926	0.9990	0.0001	0.9951	0.9985	0.0001	0.9926
normal	0.9857	0.0169	0.9954	0.9718	0.0315	0.9844	0.9665	0.0286	0.9478
satan	0.9986	0.0004	0.8212	0.9966	0.0027	0.8732	0.9963	0.0027	0.8273
ipsweep	0.9999	0.0000	0.9706	0.9999	0.0	0.9706	0.9999	0.0000	0.9706
portsweep	0.9982	0.0017	0.8927	0.9990	0.0009	0.9492	0.9984	0.0015	0.9463
nmap	1.0	0.0	1.0	0.9997	0.0003	1.0000	0.9998	0.0002	1.0000
back	0.9976	0.0	0.3798	0.9965	0.0	0.0729	0.9916	0.0074	0.7359
teardrop	0.9999	0.0001	0.6667	0.9999	0.0001	0.6667	0.9999	0.0001	0.8333
pod	0.9999	0.0	0.7471	0.9999	0.0001	0.8851	0.9999	0.0001	0.8966
guess_passwd	0.9851	0.0	0.0	0.9851	0.0	0.0	0.9851	0.0	0.0
buffer_overflow	0.9999	0.0	0.0	0.9999	0.0	0.0	0.9999	0.0	0.0
land	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
warezmaster	0.9945	0.0	0.0	0.9945	0.0	0.0	0.9945	0.0	0.0
imap	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
rootkit	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
loadmodule	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
ftp_write	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
multihop	0.9999	0.0	0.0	0.9999	0.0	0.0	0.9999	0.0	0.0
phf	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
perl	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0

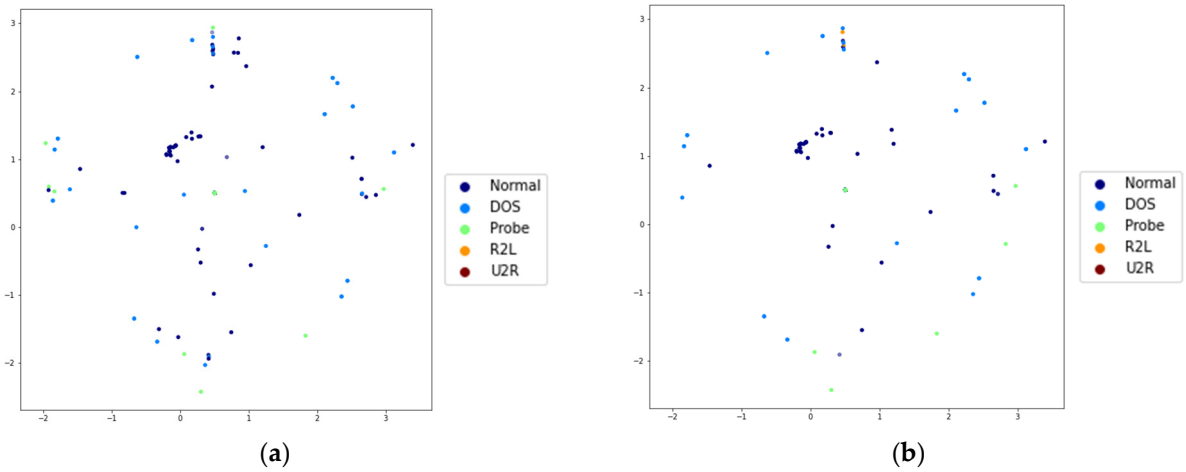
Table 3. Overall accuracy for three approaches on the KDD CUP 99 dataset.

	Max Count Approach	**Hierarchical Approach with Logistic Regression**	**Hierarchical Approach with Decision Tree**
Accuracy	0.9733	0.9707	0.9654

4.4.2. NSL-KDD Dataset

In the second experiment, we applied our method to the NSL-KDD dataset to detect four main attack types, DoS, Probe, U2R and R2L, as well as normal. As shown in Figure 6, although clustering patterns for type normal could be found in the middle of both maps, there was not a very significant clustering for the rest of the data in both the training and testing set when compared with the KDD CUP 99 result. This indicated that the NSL-KDD dataset is a more complex and challenging dataset. But, similar to KDD CUP 99, the training GSOM and testing GSOM were found to have very similar clustering results. Most of the neurons showed the same color at the same position in the training and testing map, indicating that although DGSOM might not produce a classification performance as good

as it produced on KDD CUP 99, it still had the potential to be able to classify the testing dataset to an acceptable extent, especially to identify data of type normal.



(**a**) (**b**)

Figure 6. GSOM representation of the NSL-KDD training data (**a**) and testing data (**b**). The representative color for a neuron in the map is the class type which has the highest frequency count.

We also identified that out of the total of 3328 neurons in the training GSOM, 1852 were pure neurons, and 626 neurons were not pure. The remaining 850 neurons were empty neurons. In the testing GSOM, we identified 4750 pure neurons, 140 not-pure neurons, and 1922 empty neurons. Among the 1186 pure neurons in the testing GSOM, 1057 neurons shared the same class label with the training GSOM. In those wrongly labeled testing neurons, attack type R2L is the most frequent attack type with a number of 636, which contributes to around 22% of the total number of R2L data inputs in the testing set. This indicates that there is a need to further train a separate classifier first for R2L, to assist the classification result from DGSOM.

In contrast to KDD CUP 99, we found that most of the not-pure neurons in the testing map were non-empty neurons in the training map, indicating that misclassification from empty neurons in the training GSOM might not contribute much to the final classification performance. Comparing the training and testing GSOM in Figure 6, we identified some mismatch in the node color, showing that the max-count-based approach might not work well on this dataset.

Based on the above findings and the findings from Experiment 1, which indicated that the unbalanced nature of the network intrusion dataset could result in a low detection rate, we decided to deal with the unbalanced data issue for NSL-KDD, and set the threshold of minority class as 1%, since we identified that DGSOM can provide generally good performance when the data samples are above 1% in Experiment 1. By this setting, we identified that both U2R and R2L were the minority classes, with only 995 and 52 data samples, respectively, in the training set. Thus, we trained a classifier for each of them. Since we only used a very small number of data to train the model, training was very fast, i.e., less than one minute. The rest of the training approach was the same as we applied in Experiment 1.

Table 4 shows the result of our model on the five network attack types. It could be found that the hierarchical-based approach performed better than the max count approach on the NSL-KDD dataset. While the decision-tree-based method showed better performance in detecting Probe and R2L attacks, the logistic regression method performed better in detecting U2R attacks. Overall accuracy for the five-class intrusion detection as shown in Table 5 demonstrated that the hierarchical approach with logistic regression achieved the highest accuracy. Other evaluation results including true negative rate, false negative rate, and precision can be found in Table A2 in Appendix A.

Table 4. Evaluation for three approaches on the NSL-KDD dataset.

		Normal	DoS	Probe	R2L	U2R
Max Count Approach	Detection Rate	0.97	0.8	0.67	0.07	0
	False Positive Rate	0.35	0.03	0.02	0	0
	Accuracy	0.79	0.91	0.95	0.88	1
Hierarchical Approach with Logistic Regression	Detection Rate	0.94	0.83	0.71	0.44	0.64
	False Positive Rate	0.23	0.01	0.02	0.03	0.01
	Accuracy	0.84	0.93	0.95	0.91	0.99
Hierarchical Approach with Decision Tree	Detection Rate	0.94	0.83	0.77	0.48	0.58
	False Positive Rate	0.21	0.01	0.02	0.03	0
	Accuracy	0.86	0.93	0.96	0.91	0.99

Table 5. Overall Accuracy For Three Approaches on NSL-KDD Dataset.

	Max Count Approach	Hierarchical Approach with Logistic Regression	Hierarchical Approach with Decision Tree
Accuracy	0.7638	0.8263	0.8154

We compared our method with other state-of-the art five-class intrusion detection methods based on NSL-KDD as shown in Table 6. We found that our hierarchical method was able to achieve the highest detection rate for U2R and R2L attacks, while preserving a high detection rate for the rest of the three network intrusion types. In particular, when compared with the last two methods, SiamIDS [56] and I-SiamIDS [57], which had been proven to be effective in improving the detection rate on imbalanced class dataset, our hierarchical approach with decision tree achieved the best accuracy in most of the attack types.

In order to measure the improvement of time complexity of our proposed method, we compared the processing time with the original GSOM algorithm and other SOM-based algorithms. The overall time to train a max-count-based classifier with DGSOM was 12.02 min. It took 13.33 min to train a hierarchical classifier based on decision tree and 14.02 min to train a hierarchical classifier based on a logistic regression algorithm. When we applied the original GSOM algorithm to train a classifier, the overall training time was 403.76 min. As such, all our proposed methods could lead to a speedup of more than 28 times when compared with the original GSOM method. When compared with other SOM-based algorithms applied on the NSL-KDD dataset [8], we found our method could also have a speedup of at least 4.78 times.

Table 6. Comparison with other methods on the NSL-KDD dataset.

		Normal	DoS	Probe	R2L	U2R
Our Max Count Approach	Detection rate	0.97	0.8	0.67	0.07	0
Our Hierarchical Approach with Logistic Regression	Detection rate	0.94	0.83	0.71	0.44	0.64
Our Hierarchical Approach with Decision Tree	Detection rate	0.94	0.83	0.77	0.48	0.58
BAT-MC [58]	Detection rate	0.97.	0.87	0.85	0.44	0.21
DNN [57]	Detection rate	0.97	0.76	0.62	0.04	0.25
CNN [57]	Detection rate	0.97	0.85	0.50	0.10	0.22

Table 6. *Cont.*

		Normal	DoS	Probe	R2L	U2R
XGBoost [57]	Detection rate	0.97	0.84	0.51	0.09	0.02
RNN [59]	Detection rate	0.96	0.83	0.83	0.25	0.12
LSTM [60]	Detection rate	0.93	0.73	0.67	0.06	0
SiamIDS [56]	Detection rate	0.94	0.80	0.57	0.25	0.49
I-SiamIDS [57]	Detection rate	0.89	0.86	0.77	0.32	0.50

5. Conclusions and Future Work

This paper proposed a novel method in dealing with the intrusion detection problem, especially with big data and provided a solution to scalability issues in a scenario with a large volume of network traffic. The key contributions of our research are, first, the latent representation of the large-volume network data can help us understand and visualize the network traffic patterns, as well as facilitate scalability and improve the accuracy in intrusion detection tasks; second, the three types of classifiers enable explainability in the proposed method, making the decision-making processes understandable to humans. The experimental results have shown that our proposed method is scalable, explainable, and effective even for the big imbalanced dataset, which has been the key challenge for most of the machine learning algorithms, and is suitable to apply in a cloud-computing environment.

Motivated by the results, we are interested in improving the current hierarchical method by performing comprehensive experiments to understand the influences of key parameters such as the spreading factor and number of data subsets of DGSOM and the hierarchical learning strategies. Secondly, in order to confirm the fairness and scalability of the proposed method, this approach should be compared with extensive state-of-the-art datasets, which will be explored in future studies. Additionally, this research does not consider scenarios where there are concept drifts in the data. GSOM demonstrates robustness for dynamic environments, and future work will focus on confirming this behavior for the proposed framework. Future studies will also focus on other related network traffic applications and the development of online incremental machine learning algorithms to cope with the unstationary distributions of the network data; for example, we can apply an online version of GSOM [61,62] to implement an online detection method with a change detection ability.

Author Contributions: Conceptualization, J.W., S.N. and D.A.; methodology, J.W., S.N. and D.A.; software, J.W. and T.K.; validation, J.W. and T.K.; formal analysis, J.W.; investigation, J.W., S.N. and D.A.; writing—original draft preparation, J.W. and D.A.; writing—review and editing, T.K. and D.A.; visualization, J.W.; supervision, S.N. and D.A.; project administration, S.N. and D.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. Evaluations for three approaches on the KDD CUP 99 dataset.

	Max Count Approach						Hierarchical Approach with Logistic Regression						Hierarchical Approach with Decision Tree					
Types	**TP Rate** [(1)]	**TN Rate** [(2)]	**FP Rate** [(3)]	**FN Rate** [(4)]	**Precision**	**Accuracy**	**TP Rate** [(1)]	**TN Rate** [(2)]	**FP Rate** [(3)]	**FN Rate** [(4)]	**Precision**	**Accuracy**	**TP Rate** [(1)]	**TN Rate** [(2)]	**FP Rate** [(3)]	**FN Rate** [(4)]	**Precision**	**Accuracy**
smurf	0.9999	0.9997	0.0003	0.0001	0.9998	0.9998	0.9999	0.9998	0.0002	0.0001	0.9998	0.9998	0.9999	0.9992	0.0008	0.0001	0.9994	0.9996
neptune	0.9926	0.9864	0.0136	0.0074	0.9474	0.9876	0.9951	0.9999	0.0001	0.0049	0.9997	0.9990	0.9926	0.9999	0.0001	0.0074	0.9997	0.9985
normal	0.9954	0.9831	0.0169	0.0046	0.9392	0.9857	0.9844	0.9685	0.0315	0.0156	0.8909	0.9718	0.9478	0.9714	0.0286	0.0522	0.8967	0.9665
satan	0.8212	0.9996	0.0004	0.1788	0.9229	0.9986	0.8732	0.9973	0.0027	0.1268	0.6426	0.9966	0.8273	0.9973	0.0027	0.1727	0.6322	0.9963
ipsweep	0.9706	1.0	0.0	0.0294	0.9581	0.9999	0.9706	1.0	0.0	0.0294	0.9674	0.9999	0.9706	1.0	0.0	0.0294	0.9550	0.9999
portsweep	0.8927	0.9983	0.0017	0.1073	0.3940	0.9982	0.9492	0.9991	0.0009	0.0508	0.5619	0.9990	0.9463	0.9985	0.0015	0.0537	0.4278	0.9984
Nmap	1.0	1.0	0.0	0.0000	0.9767	1.0	1.0000	0.9997	0.0003	0.0000	0.5283	0.9997	1.0000	0.9998	0.0002	0.0000	0.6563	0.9998
back	0.3798	1.0	0.0	0.6202	0.9789	0.9976	0.0729	1.0000	0.0	0.9271	0.9302	0.9965	0.7359	0.9926	0.0074	0.2641	0.2722	0.9916
teardrop	0.6667	0.9999	0.0001	0.3333	0.2000	0.9999	0.6667	0.9999	0.0001	0.3333	0.2051	0.9999	0.8333	0.9999	0.0001	0.1667	0.2381	0.9999
pod	0.7471	1.0	0.0	0.2529	0.8553	0.9999	0.8851	0.9999	0.0001	0.1149	0.8370	0.9999	0.8966	0.9999	0.0001	0.1034	0.8298	0.9999
guess_passwd	0.0	1.0	0.0	1.0	nan	0.9851	0.0	1.0	0.0	1.0	nan	0.9851	0.0	1.0	0.0	1.0	nan	0.9851
buffer_overflow	0.0	1.0	0.0	1.0	nan	0.9999	0.0	1.0	0.0	1.0	0.0	0.9999	0.0	1.0	0.0	1.0	0.0000	0.9999
land	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0
warezmaster	0.0	1.0	0.0	1.0	nan	0.9945	0.0	1.0	0.0	1.0	nan	0.9945	0.0	1.0	0.0	1.0	nan	0.9945
imap	0.0	1.	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0
rootkit	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0
loadmodule	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0
ftp_write	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0
multihop	0.0	1.0	0.0	1.0	nan	0.9999	0.0	1.0	0.0	1.0	nan	0.9999	0.0	1.0	0.0	1.0	nan	0.9999
phf	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0
perl	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan	1.0

[(1)] TP Rate: True Positive Rate; [(2)] TN Rate: True Negative Rate; [(3)] FP Rate: False Positive Rate; [(4)] FN Rate: False Negative Rate.

Table A2. Evaluations for three approaches on the NSL-KDD dataset.

		Normal	DoS	Probe	R2L	U2R
Max Count Approach	True Positive Rate	0.97	0.8	0.67	0.07	0
	True Negative Rate	0.65	0.97	0.98	1	1
	False Positive Rate	0.35	0.03	0.02	0	0
	False Negative Rate	0.03	0.2	0.33	0.93	1
	Precision	0.68	0.93	0.83	0.93	nan
	Accuracy	0.79	0.91	0.95	0.88	1
Hierarchical Approach with Logistic Regression	True Positive Rate	0.94	0.83	0.77	0.48	0.58
	True Negative Rate	0.79	0.99	0.98	0.97	1
	False Positive Rate	0.21	0.01	0.02	0.03	0
	False Negative Rate	0.06	0.17	0.23	0.52	0.42
	Precision	0.77	0.97	0.86	0.69	0.27
	Accuracy	0.86	0.93	0.96	0.91	0.99
Hierarchical Approach with Decision Tree	True Positive Rate	0.94	0.83	0.71	0.44	0.64
	True Negative Rate	0.77	0.99	0.98	0.97	0.99
	False Positive Rate	0.23	0.01	0.02	0.03	0.01
	False Negative Rate	0.06	0.17	0.29	0.56	0.36
	Precision	0.76	0.97	0.84	0.72	0.21
	Accuracy	0.84	0.93	0.95	0.91	0.99

References

1. Huang, M.Y.; Jasper, R.J.; Wicks, T.M. A Large Scale Distributed Intrusion Detection Framework Based on Attack Strategy Analysis. *Comput. Netw.* **1999**, *31*, 2465–2475. [CrossRef]
2. Obeidat, I.M.; Hamadneh, N.; Alkasassbeh, M.; Almseidin, M.; AlZubi, M.I. Intensive Preprocessing of KDD Cup 99 for Network Intrusion Classification Using Machine Learning Techniques. *Int. J. Interact. Mob. Technol.* **2018**, *13*, 70–84. [CrossRef]
3. Ortega-Fernandez, I.; Liberati, F. A Review of Denial of Service Attack and Mitigation in the Smart Grid Using Reinforcement Learning. *Energies* **2023**, *16*, 635. [CrossRef]
4. Verwoerd, T.; Hunt, R. Intrusion Detection Techniques and Approaches. *Comput. Commun.* **2002**, *25*, 1356–1365. [CrossRef]
5. Nabi, F.; Zhou, X. Enhancing Intrusion Detection Systems through Dimensionality Reduction: A Comparative Study of Machine Learning Techniques for Cyber Security. *Cyber Secur. Appl.* **2024**, *2*, 100033. [CrossRef]
6. Pradhan, M.; Kumar Pradhan, S.; Sahu, S.K. Anomaly Detection Using Artificial Neural Network. *Int. J. Eng. Sci. Emerg. Technol.* **2012**, *2*, 29–36.
7. Alkasassbeh, M.; Al-Naymat, G.; Hassanat, A.B.A.; Almseidin, M. Detecting Distributed Denial of Service Attacks Using Data Mining Techniques. *IJACSA Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 1. [CrossRef]
8. Ibrahim, L.M.; Taha, D.B.; Mahmod, M.S. A Comparison Study for Intrusion Database (KDD99, NSL-KDD) Based on Self Organization Map (SOM) Artificial Neural Network. *J. Eng. Sci. Technol.* **2013**, *8*, 107–119.
9. Abdul, A.; Vermeulen, J.; Wang, D.; Lim, B.Y.; Kankanhalli, M. Trends and Trajectories for Explainable, Accountable and Intelligible Systems: An HCI Research Agenda. In Proceedings of the Conference on Human Factors in Computing Systems-Proceedings 2018, Montreal, QC, Canada, 21–26 April 2018. [CrossRef]
10. Qu, X.; Yang, L.; Guo, K.; Ma, L.; Sun, M.; Ke, M.; Li, M. A Survey on the Development of Self-Organizing Maps for Unsupervised Intrusion Detection. *Mob. Netw. Appl.* **2021**, *26*, 808–829. [CrossRef]
11. Choksi, K.; Shah, B.; Ompriya Kale, A.; Programme, M. Intrusion Detection System Using Self Organizing Map: A Survey. *J. Eng. Res. Appl.* **2014**, *4*, 11–16.
12. Dlamini, G.; Fahim, M. DGM: A Data Generative Model to Improve Minority Class Presence in Anomaly Detection Domain. *Neural Comput. Appl.* **2021**, *33*, 13635–13646. [CrossRef]
13. Tsai, C.F.; Hsu, Y.F.; Lin, C.Y.; Lin, W.Y. Intrusion Detection by Machine Learning: A Review. *Expert Syst. Appl.* **2009**, *36*, 11994–12000. [CrossRef]
14. Tartakovsky, A.G.; Polunchenko, A.S.; Sokolov, G. Efficient Computer Network Anomaly Detection by Changepoint Detection Methods. *IEEE J. Sel. Top. Signal Process.* **2013**, *7*, 4–11. [CrossRef]

15. Ahmed, M.; Naser Mahmood, A.; Hu, J. A Survey of Network Anomaly Detection Techniques. *J. Netw. Comput. Appl.* **2016**, *60*, 19–31. [CrossRef]
16. Liao, H.J.; Richard Lin, C.H.; Lin, Y.C.; Tung, K.Y. Intrusion Detection System: A Comprehensive Review. *J. Netw. Comput. Appl.* **2013**, *36*, 16–24. [CrossRef]
17. Maciej Serda; Becker, F.G.; Cleary, M.; Team, R.M.; Holtermann, H.; The, D.; Agenda, N.; Science, P.; Sk, S.K.; Hinnebusch, R.; et al. Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey. *Uniw. Śląski* **2017**, *7*, 343–354.
18. Maind, S.B.; Wankar, P. Research Paper on Basic of Artificial Neural Network. *Int. J. Recent Innov. Trends Comput. Commun.* **2014**, *2*, 96–100.
19. Laskov, P.; Düssel, P.; Schäfer, C.; Rieck, K. Learning Intrusion Detection: Supervised or Unsupervised? *Lect. Notes Comput. Sci.* **2005**, *3617*, 50–57. [CrossRef]
20. Stolfo, S.J.; Wei, F.; Lee, W.; Prodromidis, A.; Chan, P.K. Kdd Cup Knowledge Discovery and Data Mining Competition. Available online: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed on 2 July 2024).
21. Ravipati, R.D.; Abualkibash, M. Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets—A Review Paper. *Int. J. Comput. Sci. Inf. Technol. (IJCSIT)* **2019**, *11*. [CrossRef]
22. Sapre, S.; Ahmadi, P.; Islam, K. A Robust Comparison of the KDDCup99 and NSL-KDD IoT Network Intrusion Detection Datasets Through Various Machine Learning Algorithms. *arXiv* **2019**, arXiv:1912.13204.
23. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A Detailed Analysis of the KDD CUP 99 Data Set. In Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009, Ottawa, ON, Canada, 8–10 July 2009. [CrossRef]
24. Lane, T. A Decision-Theoritic, Semi-Supervised Model for Intrusion Detection. In *Machine Learning and Data Mining for Computer Security*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 157–177. [CrossRef]
25. Aslam, J.; Bratus, S.; Pavlu, V. Semi-Supervised Data Organization for Interactive Anomaly Analysis. In Proceedings of the Proceedings-5th International Conference on Machine Learning and Applications, ICMLA 2006, Orlando, FL, USA, 14–16 December 2006; pp. 55–62. [CrossRef]
26. Chen, C.; Gong, Y.; Tian, Y. Semi-Supervised Learning Methods for Network Intrusion Detection. In Proceedings of the Conference Proceedings-IEEE International Conference on Systems, Man and Cybernetics, Singapore, 12–15 October 2008; pp. 2603–2608. [CrossRef]
27. Vinutha, H.P.; Poornima, B. Analysis of NSL-KDD Dataset Using K-Means and Canopy Clustering Algorithms Based on Distance Metrics. *Stud. Comput. Intell.* **2019**, *771*, 193–200. [CrossRef]
28. Tao, L.J.; Hong, L.Y.; Yan, H. The Improvement and Application of a K-Means Clustering Algorithm. In Proceedings of the 2016 IEEE International Conference on Cloud Computing and Big Data Analysis, ICCCBDA 2016, Chengdu, China, 5–7 July 2016; pp. 93–96. [CrossRef]
29. Shyu, M.-L.; Chen, S.-C.; Sarinnapakorn, K.; LastName, L. A Novel Anomaly Detection Scheme Based on Principal Component Classifier. In Proceedings of the Proceeding of ICDM Foundation and New Direction of Data Mining Workshop, Melbourne, FL, USA, 19–22 November 2003.
30. Zuech, R.; Khoshgoftaar, T.M.; Wald, R. Intrusion Detection and Big Heterogeneous Data: A Survey. *J. Big Data* **2015**, *2*, 1–41. [CrossRef]
31. Ippoliti, D.; Zhou, X. A-GHSOM: An Adaptive Growing Hierarchical Self Organizing Map for Network Anomaly Detection. *J. Parallel Distrib. Comput.* **2012**, *72*, 1576–1590. [CrossRef]
32. Ramadas, M.; Ostermann, S.; Tjaden, B. Detecting Anomalous Network Traffic with Self-Organizing Maps. *Lect. Notes Comput. Sci.* **2003**, *2820*, 36–54. [CrossRef]
33. Sarasamma, S.T.; Zhu, Q.A.; Huff, J. Hierarchical Kohonenen Net for Anomaly Detection in Network Security. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **2005**, *35*, 302–312. [CrossRef]
34. Gunes Kayacik, H.; Nur Zincir-Heywood, A.; Heywood, M.I. A Hierarchical SOM-Based Intrusion Detection System. *Eng. Appl. Artif. Intell.* **2007**, *20*, 439–451. [CrossRef]
35. Tan, L.; Li, C.; Xia, J.; Cao, J. Application of Self-Organizing Feature Map Neural Network Based on K-Means Clustering in Network Intrusion Detection. *Comput. Mater. Contin.* **2019**, *61*, 275–288. [CrossRef]
36. Amini, M.; Jalili, R.; Shahriari, H.R. RT-UNNID: A Practical Solution to Real-Time Network-Based Intrusion Detection Using Unsupervised Neural Networks. *Comput. Secur.* **2006**, *25*, 459–468. [CrossRef]
37. Alahakoon, D.; Halgamuge, S.K.; Srinivasan, B. Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery. *IEEE Trans. Neural Netw.* **2000**, *11*, 601–614. [CrossRef]
38. Apache Spark-Unified Engine for Large-Scale Data Analytics. Available online: https://spark.apache.org/ (accessed on 2 July 2024).
39. Gupta, G.P.; Kulariya, M. A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark. *Procedia Comput. Sci.* **2016**, *93*, 824–831. [CrossRef]
40. Jayaratne, M.; Alahakoon, D.; De Silva, D.; Yu, X. Apache Spark Based Distributed Self-Organizing Map Algorithm for Sensor Data Analysis. In Proceedings of the Proceedings IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society 2017, Beijing, China, 29 October–1 November 2017; pp. 8343–8349. [CrossRef]

41. Beyer, K.; Goldstein, J.; Ramakrishnan, R.; Shaft, U. When Is "Nearest Neighbor" Meaningful? In Proceedings of the Database Theory—ICDT'99: 7th International Conference Proceedings 7, Jerusalem, Israel, 10 January 1999; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1540, pp. 217–235.
42. Zhang, T.; Wang, X.; Li, Z.; Guo, F.; Ma, Y.; Chen, W. A Survey of Network Anomaly Visualization. *Sci. China Inf. Sci.* **2017**, *60*, 1–17. [CrossRef]
43. Kohonen, T. The Self-Organizing Map. *Proc. IEEE* **1990**, *78*, 1464–1480. [CrossRef]
44. Jayaratne, K.M.C. Multimodal Perceptual Mechanisms for Unsupervised Self-Structuring Artificial Intelligence in Distributed Systems. Doctoral dissertation, La Trobe University, Melbourne, Australia, 2020.
45. Banković, Z.; Moya, J.M.; Araujo, Á.; Fraga, D.; Vallejo, J.C.; De Goyeneche, J.M. Distributed Intrusion Detection System for Wireless Sensor Networks Based on a Reputation System Coupled with Kernel Self-Organizing Maps. *Integr. Comput.-Aided Eng.* **2010**, *17*, 87–102. [CrossRef]
46. Sammon, J.W. A Nonlinear Mapping for Data Structure Analysis. *IEEE Trans. Comput.* **1969**, *C-18*, 401–409. [CrossRef]
47. Yin, S.; Li, H.; Sun, Y.; Ibrar, M.; Teng, L. Data Visualization Analysis Based on Explainable Artificial Intelligence: A Survey. *IJLAI Trans. Sci. Eng.* **2024**, *2*, 13–20.
48. Chatti, M.A.; Guesmi, M.; Muslim, A. Visualization for Recommendation Explainability: A Survey and New Perspectives. *ACM Trans. Interact. Intell. Syst.* **2024**, *14*, 1–40. [CrossRef]
49. Ables, J.; Kirby, T.; Anderson, W.; Mittal, S.; Rahimi, S.; Banicescu, I.; Seale, M. Creating an Explainable Intrusion Detection System Using Self Organizing Maps. In Proceedings of the 2022 IEEE Symposium Series on Computational Intelligence, SSCI 2022, Singapore, 4–7 December 2022; pp. 404–412. [CrossRef]
50. Kirby, T.M.; Rahimi, S.; Mittal, S.; Banicescu, I.; Perkins, A.; Jankun-Kelly, T.J.; Keith, J.M. Pruning GHSOM to Create an Explainable Intrusion Detection System. Master Thesis, Mississippi State University, Starkville, MS, USA, 2023.
51. Wickramasinghe, C.S.; Amarasinghe, K.; Marino, D.L.; Rieger, C.; Manic, M. Explainable Unsupervised Machine Learning for Cyber-Physical Systems. *IEEE Access* **2021**, *9*, 131824–131843. [CrossRef]
52. Mahbooba, B.; Timilsina, M.; Sahal, R.; Serrano, M. Explainable Artificial Intelligence (XAI) to Enhance Trust Management in Intrusion Detection Systems Using Decision Tree Model. *Complexity* **2021**, *2021*, 6634811. [CrossRef]
53. Lippmann, R.P.; Fried, D.J.; Graf, I.; Haines, J.W.; Kendall, K.R.; McClung, D.; Weber, D.; Webster, S.E.; Wyschogrod, D.; Cunningham, R.K.; et al. Evaluating Intrusion Detection Systems: The 1998 DARPA off-Line Intrusion Detection Evaluation. In Proceedings of the Proceedings-DARPA Information Survivability Conference and Exposition, DISCEX 2000, Hilton Head, SC, USA, 25–27 January 2000; Volume 2, pp. 12–26. [CrossRef]
54. Ingre, B.; Yadav, A. Performance Analysis of NSL-KDD Dataset Using ANN. In Proceedings of the International Conference on Signal Processing and Communication Engineering Systems-Proceedings of SPACES 2015, in Association with IEEE, Guntur, India, 2–3 January 2015; pp. 92–96. [CrossRef]
55. Özgür Corresp, A.; Erdem, H.; OzgürOzg, A. The Impact of Using Large Training Data Set KDD99 on Classification Accuracy. *PeerJ Prepr.* **2017**, *5*, e2838. [CrossRef]
56. Bedi, P.; Gupta, N.; Jindal, V. Siam-IDS: Handling Class Imbalance Problem in Intrusion Detection Systems Using Siamese Neural Network. *Procedia Comput. Sci.* **2020**, *171*, 780–789. [CrossRef]
57. Bedi, P.; Gupta, N.; Jindal, V. I-SiamIDS: An Improved Siam-IDS for Handling Class Imbalance in Network-Based Intrusion Detection Systems. *Appl. Intell.* **2021**, *51*, 1133–1151. [CrossRef]
58. Su, T.; Sun, H.; Zhu, J.; Wang, S.; Li, Y. BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset. *IEEE Access* **2020**, *8*, 29575–29585. [CrossRef]
59. Yin, C.; Zhu, Y.; Fei, J.; He, X. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. *IEEE Access* **2017**, *5*, 21954–21961. [CrossRef]
60. Ding, Y.; Zhai, Y. Intrusion Detection System for NSL-KDD Dataset Using Convolutional Neural Networks. *ACM Int. Conf. Proceeding Ser.* **2018**, *2*, 81–85. [CrossRef]
61. Kempitiya, T.; Alahakoon, D.; Osipov, E.; Kahawala, S.; De Silva, D. A Two-Layer Self-Organizing Map with Vector Symbolic Architecture for Spatiotemporal Sequence Learning and Prediction. *Biomimetics* **2024**, *9*, 175. [CrossRef]
62. Nallaperuma, D.; Nawaratne, R.; Bandaragoda, T.; Adikari, A.; Nguyen, S.; Kempitiya, T.; De Silva, D.; Alahakoon, D.; Pothuhera, D. Online Incremental Machine Learning Platform for Big Data-Driven Smart Traffic Management. *IEEE Trans. Intell. Transp. Syst.* **2019**, *20*, 4679–4690. [CrossRef]