A Fast and Effective Method for Intrusion Detection using Multi-Layered Deep Learning Networks

A. Srikrishnan[1], Dr. Arun Raaza[2], Dr. Ebenezer Abishek. B[3]
Dr. V. Rajendran[4], Dr. M. Anand[5], S. Gopalakrishnan[6], Dr. Meena. M[7]
Research Scholar, Department of ECE, Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai, India[1]
Deputy Director of CARD, Department of ECE, Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai, India[2]
Associate Professor, Department of ECE Vel Tech Multi Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Chennai, India[3]
Director and Professor, Department of ECE, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai, India[4]
Professor, Department of ECE, Dr.M.G.R. Educational and Research Institute, Chennai, India[5]
Assistant Professor, Department of MEE, Sengunthar College of Engineering, Tiruchengode[6]
Associate Professor, Department of ECE, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai, India[7]

Abstract*—The practise of recognising unauthorised abnormal actions on computer systems is referred to as intrusion detection. The primary goal of an Intrusion Detection System (IDS) is to identify user behaviours as normal or abnormal based on the data they communicate. Firewalls, data encryption, and authentication techniques were all employed in traditional security systems. Current intrusion scenarios, on the other hand, are very complex and capable of readily breaching the security measures provided by previous protection systems. However, current intrusion scenarios are highly sophisticated and are capable of easily breaking the security mechanisms imposed by the traditional protection systems. Detecting intrusions is a challenging aspect especially in networked environments, as the system designed for such a scenario should be able to handle the huge volume and velocity associated with the domain. This research presents three models, APID (Adaptive Parallelized Intrusion Detection), HBM (Heterogeneous Bagging Model) and MLDN (Multi Layered Deep learning Network) that can be used for fast and efficient detection of intrusions in networked environments. The deep learning model has been constructed using the Keras library. The training data is preprocessed and segregated to fit the processing architecture of neural networks. The network is constructed with multiple layers and the other required parameters for the network are set in accordance with the input data. The trained model is validated using the validation data that has been specifically segregated for this purpose.

Keywords*—Intrusion detection system; knowledge discovery and data mining; transmission control protocol; adaptive parallelized intrusion detection; constrained-optimization-based extreme learning machine

I. Introduction

IDS models can serve a wide range of purposes and requirements when applied in business settings. One of the most popular applications is the method of intrusion detection in personal systems or distributed settings [1]. The design of modern operating systems includes the implementation of technology that detect and prevent intrusions. However, the handling capabilities of these systems are currently unknown. As a consequence of this, the majority of customers choose to invest in expert intrusion detection solutions for enhanced levels of protection. In addition, there is a considerable need for IDS that may be implemented in clustered systems and used in servers [2, 3]. There are many commercially available intrusion detection systems, some of which include the Bro intrusion detection system, which was developed by VISTAS Labs and the School of Engineering, the Snort intrusion detection system, which is distributed under the GNU licence [4], Network Protocol Analyzer [6], Multi Router Traffic Grapher (MRTG) [7], and a few other options. On the other hand, the computing requirements of the majority of these systems, as well as their accuracy, might be enhanced.

A. Motivation of this Research

This research was motivated by the fact that the majority of currently available intrusion detection systems do not handle the issues listed above as part of their operational process. As a result, there is a need to design an effective intrusion detection system with mechanisms to handle data imbalance and concept drift while simultaneously achieving better accuracy and faster detection of intrusions. This research was carried out to fulfil this need.

B. Objectives

The primary purpose of this work is to create an efficient intrusion detection system that is capable of the identification of intrusion signatures in network data in real time. The secondary goals are as follows:

- To develop a method of intrusion detection that is capable of dealing with the inherent data imbalance that is present within the domain in an efficient manner.
- In order to effectively deal with concept drift, which is an essential component of the domain.

- To incorporate feature reduction in order to lessen the demands placed on the model's computational resources.
- To enable quicker detection of intrusions, parallelization is going to have to be incorporated into the detection process.
- To execute detection of intrusions in real time in order to minimise financial damage as much as possible.

II. Literature Survey

One of the essential components of today's continuously networked world is the presence of an intrusion detection system. As a result, there have been a few examination promises made in this area. In this section, we will discuss what are the most recent commitments in the field of intrusion detection. The investigation of the models will be carried out in three important stages. "The primary section discusses the function of AI-based models in identifying intrusions, the subsequent section investigates the significance of component choice in this space, and the concluding section discusses the function of adaptable models in the field of intrusion detection. H. Yang et al. [8] presented a novel and factual model that makes use of Least Square Help Vector Machines (LS-SVM) in order to recognise intrusions. This model divides the data into subgroups that are not consistent with one another. A delegate test will be selected from within this subgroup in order to prepare the model. SVM-based intrusion detection systems are incorporated into another comparative SVM-based model for network intrusion detection models [9]. The most important AI methods have been implemented in a number of the models, and those models also demonstrate intriguing expectations. These models include hereditary and fluffy calculation based models [10], grouping and k-closest neighbour based models [11], IDS utilising Backing Vector Machines (SVM) for preparation [12, 13], and a lot of other similar models. These models additionally rely on signature-based detection of intrusions in order to function well. In most cases, they are made as twofold classifiers, and they are prepared on both typical and irregular marks. Typical marks are the ones that are used. It was determined that the models were computationally incomprehensible, which resulted in significant time requirements. A model of IDS with several layers was suggested [14]. This is a component choice based model, which considers attacks to be layers and selects highlights for each of the layers in order to construct the detection model. It was proposed [15] to use an ongoing-based irregularity detection model for the purpose of network intrusion detection. Keeping up with adaptive mark databases that are not difficult to renew and reproduce under continuous settings is essential to this concept. In addition to this, the model suggests a multi-objective component determination method for the practical selection of attributes that will result in increased levels of precision. An earlier version of this model that deals with the detection of intrusions on systems that have been implanted was proposed [16]. The study [17] presents a proposal for a staggered intrusion detection paradigm that is based on peculiarity detection. A comparable methodology for the detection of peculiarities based on trees was suggested [18]. In order to identify intrusions, this model relies on a combination of calculations based on the Firefly and Hereditary algorithms. There was a proposal made for a grouping-based intrusion detection approach [19]. The Semi-Directed Multifaceted Grouping Model (SMLC) that was proposed in this work makes use of named data in some capacity for the preparation process, which enables an adjustable detection procedure. Other semi-directed intrusion detection models include a group-based IDS Al- [21], a normal neighbour based model [22], and a semi-regulated model [20].

The models make use of fundamental techniques, which leads to diminished performance when applied to unbalanced data. The author [23] made a suggestion for a model that handled imbalances in the IDS. This is a real-time model that is based on clustering and use the RIPPER algorithm for the detection procedure. The research [24] presented a model with a similar structure that was based on the RIPPER algorithm. The study [25] presented an idea for an intrusion detection model that was based on principal component analysis (PCA). This paradigm is a profiling-based one, and it constructs profiles by making use of the intrusion signatures. Because this model is based on several classifications of classes, it is intended to recognise a wide variety of intrusion signs so that it can provide accurate categorizations. The study [26] presented a proposal that included an in-depth investigation of the classification models that can be utilised for intrusion detection in the most efficient manner.

An efficient methodology for the selection of features has been proposed [27] for use with the KDD CUP 99 dataset. The programme was able to make fairly accurate predictions despite having only six characteristics to work with. In a similar vein, the Flexible Neural Trees model [28] was able to attain an accuracy of 99.19% with just four features. A model for the identification of intrusions that was written in A C# was suggested [29]. Using this method results in the creation of a packet sniffer that has the capacity to effectively gather packets from an interactive TCP session and inspect them. Attackers will frequently engage in packet chaffing whenever they are dealing with models of packet sniffers. Researchers can more easily identify these types of packets with the help of the model, which works by injecting more packets into the network. The study [30] presented a strategy for identifying stepping stone intruders in their research. This model is responsible for carrying out the process of intrusion detection by contrasting the contents of a host's incoming and outgoing traffic packets. When the contents are examined, the model is evaluated to determine whether or not it can serve as a stepping stone. Because of this, it is possible to make a clear distinction between a typical packet and an invasive packet. On the other hand, if the packets are encrypted, this paradigm for detecting intrusions may not be successful. Because of this, inspecting packets cannot be called a model that is 100 percent reliable for spotting invasions. The research [31] presented a model for the identification of intrusions that was very comparable. The stepping stone attack has also been suggested as being detectable by using this concept. In contrast, this model identifies the packet source based on the timestamp, size, and sequence number of the data packets rather than by inspecting the contents of the packets themselves. Even after they have been encrypted, these parameters are still legible; hence, this model is seen to have superior performance when compared to the model that came before it. Examining the information included within the packet header was also the subject of a suggestion made by [32]. Both [33] and [34] presented an additional method that makes use of the information regarding the packet count in order to identify the stepping stone assault. When it comes to fending off stepping stone attacks, connection chain difficulties are seen as being among the most important components. The author [35] presented a model that determines the stepping stone attack by estimating the length of the connecting chain as a starting point for the calculation. The study [36] presented a model with the aim of precisely determining the connecting chain more of the time. Even while these methods assist cut down on the amount of time needed for training and detection, because they are unable to properly deal with concept drift, they are not ideal for real-time intrusion detection.

III. Methodology

Detecting an intrusion into a system typically entails searching through a vast repository for intrusion signatures that are particularly sophisticated. For this purpose, a complicated model that recognises these signatures is required. This research provides a neural network model that is based on deep learning and has the capability of performing efficient intrusion detection on network transmission data. The Multi Layered Deep Learning Network that has been suggested is a deep learning network since it is made up of a number of hidden processing layers at various depths across the network. It was discovered that detection through the use of the deep network exhibited effective performances when it came to detecting the intrusion signatures.

A. *Multi-Layered Deep Learning Networks (MLDN)*

In this paper, a Multi-Layer Deep Learning Network (MLDN) model is presented for the detection of intrusions in a timely and accurate manner. The neural network is one of the primary models that is utilised in the quest to make accurate forecasts. Deep network intrusion detection is capable of identifying a great number of intrinsic patterns, in addition to addressing issues of data imbalance and concept drift. The deep learning architecture that has been proposed for use in intrusion detection is broken down into four distinct stages. These stages are data pre-processing, data separation, network development, and model fitting. Fig. 1 illustrates the suggested model's design, and the corresponding pseudocode may be found below.

Architecture of the Algorithm for the MLDN:

1) Input transaction data.

2) Perform data pre-processing to eliminate inconsistencies.

3) Separate the data into three categories: training, testing, and validation.

4) Construct the neural network using the input data as a basis.

a) Create Layer and assign activation function.

b) Determine Epoch.

c) Assign Learning rate.

d) Assign Optimizer and Loss function.

e) Data Shuffling.

f) Create Layer and assign activation function b.

5) Get the process of network training off the ground.

6) Validate the trained model by using the data from the validation.

7) Using the results of the tests, determine the final forecast.

B. *Data Collection*

The model has been validated through the utilisation of industry-standard benchmark datasets such as NSL-KDD, KDD CUP 99, and Koyoto 2022+ datasets.

C. *Data Pre-Processing Phase*

In most cases, neural networks are unable to effectively manage all of the different types of data that are incorporated into network data. They are only able to deal with data of the double type, and they require all of the data to fall within the same range in order for it to be relevant to both sets of characteristics. These criteria are dealt with during the pre-processing phase. The procedures that were carried out during the pre-processing phase are illustrated in Fig. 2. Data normalisation follows data imputation. One of the necessary pre-processing steps that must be completed before working with real-time data is known as data normalisation. The operational nature of machine learning models makes it necessary to implement a significant amount of standardisation.

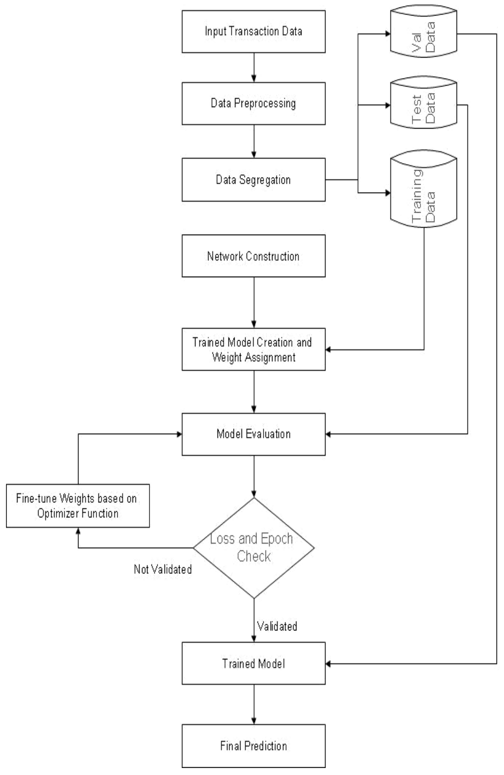

Fig. 1. Proposed MLDN architecture.

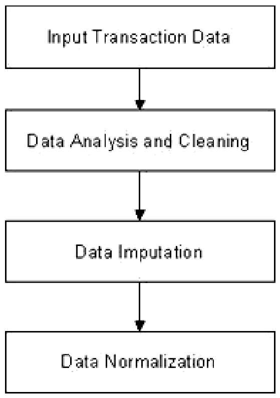

Fig. 2. Data pre-processing.

Every machine learning model has the tendency to fit functions to the data that is provided. In most cases, the weights that are applied to an attribute are what decide the level of relevance that the attribute has. The process of fitting the functions has a tendency to become more difficult when the training data comprises values that fall within a wide range of values. If any of the attributes include significant values, this will ultimately result in the actual value being the one to determine the significance of the variable. Because of this, the weights that were assigned are likely to be useless. As a consequence of this, it becomes vital to transform all of the data into comparable ranges so that consistency can be achieved throughout the process of prediction. The three normalising techniques that are employed the most frequently and extensively are the min-max normalisation, the z-score normalisation, and the decimal scaling. The original data range is transformed in a linear fashion with the application of the Min-Max normalisation. This is demonstrated by:

$$x' = \left(\frac{x - \min(A)}{\max(A) - \min(A)}\right) * (D - C) + C \quad (1)$$

where "x" represents the normalised value and "x" represents the actual value of the attribute A. The data will be scaled between the predetermined limits [C, D], which are denoted by the letters C and D. Another method that can be utilised in the normalisation process is known as the Z-score normalisation method. The data are normalised between the intervals of 0 and 1 using this model. This follows logically from the formula.

$$x_i' = \frac{x_i - \bar{A}}{std\ (A)} \quad (2)$$

where xi' and xi are the normalised and actual values of the attribute A, A is the mean value for the attribute A, and std(A) is given by where xi' and xi are the normalised and actual values of the attribute A.

$$std(A) = \sqrt{\frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \bar{A})^2} \quad (3)$$

where "n" refers to the total number of rows or instances contained inside the data. The decimal scaling approach is the simplest one, and it yields results that are dependent on both the current value and the highest value that can be found in the property. This is demonstrated by

$$x' = \frac{x}{10^j} \quad (4)$$

If x' represents the normalised value, x represents the actual value of the attribute A, and j represents the number of digits that make up the highest possible number in the variable A.

For the purpose of normalisation, this study makes use of Min-Max Normalization because it provides the benefit of being able to set both the minimum and the maximum values.

D. Data Segregation Phase

Now that the data have been standardised, the models may be trained using them. On the other hand, it is essential to keep in mind that model validation is a requirement that must be met by any machine learning model. Because of this, the normalised training data is separated into three distinct components: the training data, the testing data, and the validation data. The data is divided in accordance with the proportions 7:2:1. Seventy percent of the total data set is used to sample the training set, twenty percent of the total data set is used to sample the test set, and ten percent of the total data set is used for validation purposes. After the data have been separated, the training data will be utilised in order to construct the trained model.

E. Network Construction Phase

During this phase, a deep neural network model is utilised to facilitate the efficient identification of intrusion signatures derived from the transmission data. In order to construct the neural network, the deep learning library known as Keras was utilised. A neural network, often called an artificial neural network, is a network of neurons that collaborates to perform effective machine learning. Neural networks are also sometimes referred to by its other name, natural neural networks. Neurons, often referred to as perception, are the individual processing pieces that are used to construct neural networks. As can be seen in Fig. 3, a single neuron has numerous inputs coming into it, but it only produces a single output.

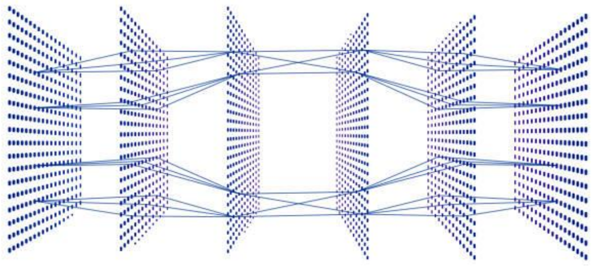

Fig. 3. Neuron: A view.

On the other hand, these inputs cannot be independently acted upon. As a consequence of this, the relative significance of each input is reflected in the weights that are assigned to it. Real numbers (w1, w2,...) are the typical notation used to express weights. The output of a neuron is often a weighted aggregate of the neuron's input value and the weights that accompany that value. This is demonstrated by

$$Output = \varphi \sum_{i=1}^{n} w_i\, x_i \quad (5)$$

where "the activation function, *w* and *x* are the weights and inputs of the neuron", and the symbol for the activation function. An input layer, one or two processing or hidden layers, and an output layer are the typical layers that make up a neural network. In most cases, the network also contains an output layer. Each layer is made up of multiple neurons, each of which is responsible for processing the information received in that layer and producing the appropriate outputs. Fig. 4 is an illustration that provides a general representation of a neural network.

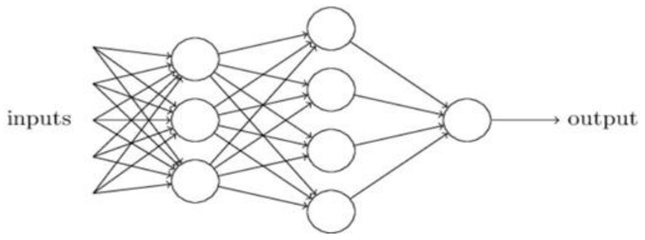

Fig. 4. A simple neural network.

Every layer in the network operates based on the input that was provided by the layer that came before it, carries out operations as specified by Eq. (5), and then passes along its output to the layer that comes after it. One variety of artificial neural network is known as a deep neural network. This network has several hidden layers within its structure. The technique of running a deep neural network is identical to the process of running an artificial neural network; however, the addition of more layers results in the provision of improved prediction capabilities on vast and difficult situations. They are typically utilised in fields that call for the analysis of massive amounts of complex data that contain a number of properties. Because the field of network intrusion detection entails the processing of massive volumes of data with complicated patterns, a deep neural network would be an effective solution to the issue. Keras is a deep learning package that is open source and based on the programming language Python. It makes it possible to create neural networks. The fact that Keras is a high-level library means that it can simply and efficiently integrate with a number of different low-level systems. This is the primary benefit of using Keras. In its current state, Keras is compatible with TensorFlow, Theano, MXNet, and Microsoft Cognitive Toolkit. Keras is typically combined with low-level base libraries like as TensorFlow and Theano. These are the two most popular low-level base libraries. The fact that Keras was designed from the ground up to be extensible, modulatory, and minimalistic is undoubtedly its most significant selling point. When used with TensorFlow, Graphics Processing Units (GPUs) can also be used in conjunction with Tensor Processing Units (TPUs) for improved and more rapid processing. This is yet another advantage of the Keras framework, which was developed with the facilities to include GPUs from the start.

Both the sequential API and the functional API can be used to construct Keras models. The sequential API is the more traditional method. Models can be built layer by layer using the sequential application programming interface. This type is suitable for the vast majority of the applications that are now available. Nevertheless, the approach does not perform very well when used to applications that share layers or that have several inputs or outputs. The functional API makes it possible to create networks that are more adaptable and diverse. It enables connections with layers at any level, which in turn makes it possible to create networks with a greater degree of complexity. Image processing, audio and video processing, and natural language processing are just few of the applications that typically make use of them. To construct the network, this work makes use of the sequential Application Programming Interface.

The neural network model is constructed with the help of the sequential application programming interface. An input layer, several hidden layers, and an output layer make up the network. Each of these levels is sandwiched between two other layers. In this particular investigation, the learning rate is 0.3. The rate at which the model must advance in order to become closer and closer to the correct response is known as the learning rate. A slower rate of convergence is indicated by smaller numbers, while a faster rate of convergence is indicated by larger values. Larger values may cause the model to bypass the optimal solution. As a consequence of this, it is necessary to identify the option that offers the highest value taking into account the circumstances. At this time, level 50 has been selected for the period. Epochs are the intervals of time during which the neural network model is provided with training data so that it can acquire new skills. Higher epochs produce better models. On the other hand, extreme care needs to be taken to prevent the model from being overfit. The parameters that were utilised are detailed in Table I.

TABLE I. NEURAL NETWORK PARAMETERS

Parameter	Value
Network Type	Sequential
Batch Size	64
Epochs	50
Learning Rate	0.1
Shuffle	True
Validation Data	Provided
Optimizer	Adam

The data that is being sent to the neural network has been shuffled to ensure that it is not arranged in any particular way before it is sent there. The network will benefit from this in the form of generic training. The validation data has been added to the network in order to ensure accurate prediction and also to prevent the network model from being overly tailored to the data. In order to perform the process of iteratively adjusting the weights of the neural network model depending on the data, an Optimizer algorithm is required to be utilised. The Adam optimizer is utilised right here. The Stochastic Gradient Descent algorithm has been expanded upon in order to create the Adam optimizer. The learning rate determines the level of update that is available. In order to generate adaptable learning rates, the algorithm modifies the levels of the learning rate at increasingly frequent intervals throughout the training process. This helps in better identifying the best possible solution to the problem. As a result, the Adam optimizer continues to be one of the optimizer algorithms that is utilised the most in neural networks.

The neural network that is suggested will have different layer configurations constructed into it depending on the dataset that is being analysed. It is intended for each of these layers to have a substantial thickness. When all of a layer's nodes are connected to all of the nodes of the layer that follows it, we refer to that layer as dense. This contributes to the construction of a network that broadcasts all of its discoveries to every accessible node on the network.

The input layer is the first layer that is created. The total number of attributes that are included in the training data is typically used to determine how many nodes should be present in the input layer. The model that has been proposed is made up of two discrete levels. Multiple neurons are incorporated into the design of the hidden layers. In this particular experiment, the successive layers each make use of 100 and 50 neurons. One neuron is present in the output layer that comes last. Since the issue that is being worked on is a binary classification issue, it would be sufficient to use a single neuron that was programmed with the output probability.

In addition to these qualities, activation functions are an extremely important factor in determining the effectiveness of the neural network. The activation function of a node in a neural network is what decides what the output of that node will be given the set of inputs for that node. This output is used as an input by the node that is located on the layer below. In most cases, the value that is produced by an activation function falls somewhere in the range of 0 to 1 or -1 to 1.

The activation function that is used is what determines the actual output that is produced. Other activation functions are available, however the sigmoid (Shown in Fig. 5), hyperbolic tangent (tanh), Rectified Linear Units (ReLU), and linear activation functions are the most frequent ones used in neural networks. Other activation functions are also available. In most cases, the sigmoid activation function has the form of the equation below:

$$f(x) = \frac{1}{1+\exp(-x)} \tag{6}$$

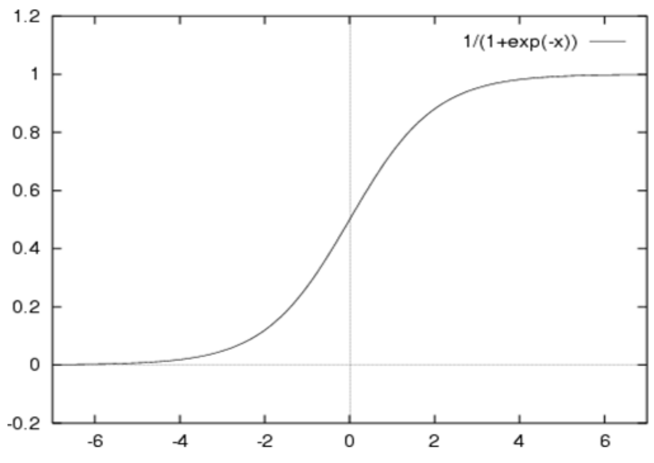

Fig. 5. Sigmoid activation function.

The range of the curve, which is S-shaped, is between 0 and 1, and the range of the curve itself is between 0 and 1. The fact that this function's range is [0, 1], which makes optimization more difficult, is the function's primary drawback. Because of its slow convergence, it is particularly well-suited for issues involving binary categorization. It has a problem with the gradient disappearing into nothingness. The activation function of the hyperbolic tangent, abbreviated as tanh, takes the form

$$f(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)} \tag{7}$$

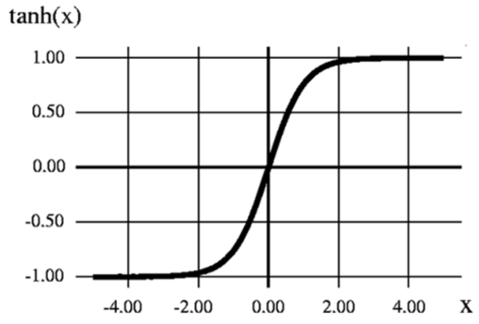

Fig. 6. TanH activation function.

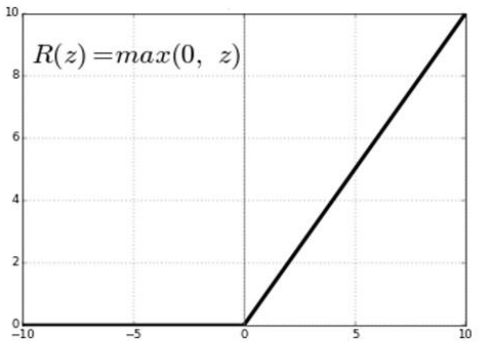

Fig. 7. ReLU activation function.

The ReLU function ranges between 0 and 1 (Fig. 7). It provides six times better convergence compared to TanH. The model is recommended for usage in intermediate layers, as it might lead to dead neurons if the input contains negative values. In the proposed approach the input and hidden layers use ReLU activation function, while the output layer uses linear activation. The S curve can also be seen in this function's plot. On the other hand, the output values are in the range of -1 to 1. This makes it much simpler to perform optimizations. However, this function also has a problem called vanishing gradient, which makes it difficult to evaluate (Fig. 6).

The phenomenon known as the "vanishing gradient problem" typically occurs in models such as neural networks that permit the backpropagation of errors. The fact that errors are typically calculated in the final output layer is the most significant problem. As a result, the layer that comes immediately before the final layer is responsible for handling the faults that have the greatest impact. However, despite the fact that faults are passed on to early layers, each layer is responsible for handling problems and only passes on errors that are still present to subsequent layers. As a consequence of this, the earliest layers do not typically get a significant amount of influence from the faults. Because of this, the early layers are the ones that take the most time to train. On the other hand, the early layers are the ones in charge of recognising fundamental patterns, which serve as the fundamental constituents of the neural network model. The whole neural networks model converges more slowly as a consequence of this issue. This problem was addressed with the development of the ReLU function. This is the form it takes.

F. Model Fitting Phase

The building of a functional model or architecture is required before the neural network can be created in the phase that came before this one. Fig. 8 depicts the order in which an artificial neural network model is developed and put into operation. The process of model fitting is the activity that actually carries out the training of the network. Before moving on to the next step, the data is first partitioned into two distinct sections: the data section and the labels section. The data part contains the attributes, with the exception of the class attribute; the class attribute is located in the labels section. The data from the training session are input into the neural network model so that it can perform the necessary analysis. The epoch value determines the total number of training iterations that the backpropagation network undergoes before being considered fully trained. The training continues until an error rate that meets the requirements is achieved. The data that needs to be forecasted is sent over this network, and then the conclusions drawn from those predictions are obtained.

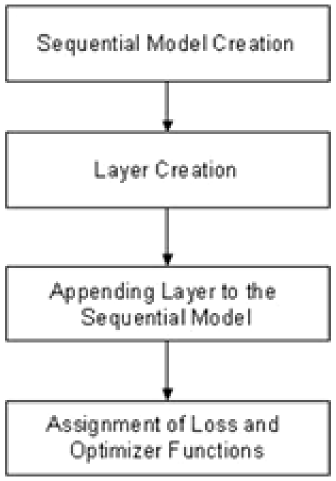
Sequential Model Creation
Layer Creation
Appending Layer to the Sequential Model
Assignment of Loss and Optimizer Functions

Fig. 8. Neural network operational sequence.

IV. Results and Discussion

Python and the Keras library suite were used to create the MLDN architecture that has been presented. The model has been validated through the utilisation of industry-standard benchmark datasets such as NSL-KDD, KDD CUP 99, and Koyoto 2022+ datasets. The network is constructed with the help of Sequential API, and then the layers that make up the neural network are added. To properly analyse each dataset, a unique network architecture will need to be developed. Tables II, III, and IV present the structures that were developed for each of the datasets that were utilised.

TABLE II. Neural Network Configuration for NSL-KDD

Layer (Type)	Activation Function	Input Dimension	Output Shape	No. of Parameters
Input (Dense)	Linear	41	(None,80)	3360
Processing 1 (Dense)	ReLU	80	(None,100)	8100
Processing 2 (Dense)	ReLU	100	(None,50)	5050
Processing 3 (Dense)	ReLU	50	(None,50)	1530
Output (Dense)	Linear	30	(None,1)	31
Total No. of Parameters				18,071
Trainable Parameters				18,071
Non-trainable Parameters				0

TABLE III. Neural Network Configuration for KDD CUP 99

Layer (Type)	Activation Function	Input Dimension	Output Shape	No. of Parameters
Input (Dense)	Linear	38	(None,50)	1950
Processing 1 (Dense)	ReLU	50	(None,250)	12,750
Processing 2 (Dense)	ReLU	250	(None,100)	25,100
Processing 3 (Dense)	ReLU	100	(None,50)	5050
Output (Dense)	Linear	50	(None,1)	51
Total No. of Parameters				44,901
Trainable Parameters				44,901
Non-trainable Parameters				0

TABLE IV. Neural Network Configuration for Koyoto 2022+

Layer (Type)	Activation Function	Input Dimension	Output Shape	No. of Parameters
Input (Dense)	Linear	18	(None,50)	950
Processing 1(Dense)	ReLU	50	(None,200)	10200
Processing 2(Dense)	ReLU	200	(None,100)	20220
Processing 3(Dense)	ReLU	100	(None,20)	2020
Output (Dense)	Linear	20	(None,1)	21
Total No. of Parameters				33,291
Trainable Parameters				33,291
Non-trainable Parameters				0

Fig. 9 presents the results of an evaluation of how well the suggested MLDN model performed on the NSL-KDD, KDD CUP 99, and Koyoto 2022+ datasets in terms of their respective ROC charts. The False Positive Rate (FPR) is represented by the x-axis, and the True Positive Rate is represented by the y-axis in this graph (TPR). It is anticipated that an effective model will demonstrate high levels of TPR while exhibiting low levels of FPR. After the points have been plotted, the graph demonstrates that the suggested model has ROC curves that are located at the (0, 1) or top-right position. This indicates that the proposed model is effective. Fig. 10 depicts the PR curve, which represents the accuracy and recall levels of the proposed model over all three datasets. An efficient model should display high values on the x-axis for recall and also display high values on the y-axis for precision (y-axis). The graph that represents the PR plot demonstrates that all three datasets have high levels of accuracy and recall, which demonstrates that the suggested model has a high level of prediction performance.

On the NSL-KDD, KDD CUP99, and Koyoto 2022+ datasets, the values that were obtained for a variety of performance measures such as FPR, TPR, Recall, and Precision, among others, are provided in Table V. The MLDN model that was proposed has levels of TPR and Precision that are extremely high, which is an indication of its effectiveness in predicting intrusion signs. In a similar vein, a high TNR level is indicative of the fact that the presented model demonstrates excellent prediction efficiency when it comes to anticipating normal transmission signals. In a similar vein, low FPR and FNR levels of less than one percent imply that the model has exceptionally low levels of incorrect predictions. As a result of this, it is abundantly clear that the MLDN model that was proposed is efficient and offers good performance.

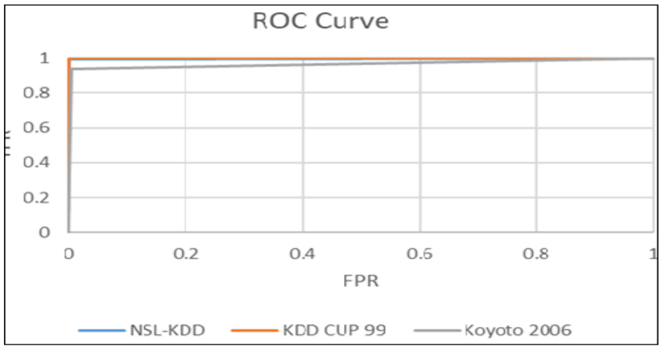

Fig. 9. ROC curve comparison of MLDN.

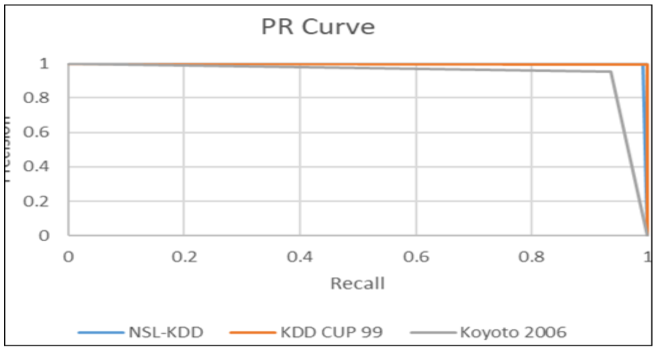

Fig. 10. PR curve comparison of MLDN.

Alterations were made to the settings of the parameters, and a sensitivity analysis was carried out on each of the three datasets in order to determine how the learning rate and the number of epochs affected the results. During the study, multiple parameter pairs were employed, and the accuracy that was acquired for each parameter combination was used during the analysis.

Table VI contains the acquired results for perusal. Within the first five sets, the learning rate is manipulated while the epoch is held constant (P1 to P5). It was possible to see that, as the learning rate was decreased, the performance on all tree datasets tended to decline to some degree, and this tendency increased as the learning rate was decreased further (P1 and P2). The learning rate is decreased, and as a result, the model takes increasingly minute steps in the direction of the best answer. As a result, 50 epochs were insufficient to accomplish the goal of achieving convergence. Taking the example of P11, where the number of epochs is increased, demonstrates that the model was able to reach convergence. When the learning rate is increased, such as in P4 and P5, the results demonstrate a decrease in performance.

TABLE V. PERFORMANCE ANALYSIS OF MLDN

Measures	NSL-KDD	KDD CUP 99	Koyoto 2022+
FPR	0.001934	0.001254	0.005605
TPR	0.991718	0.998519	0.93578
Recall	0.991718	0.998519	0.93578
Precision	0.997917	0.995079	0.953271
TNR	0.998066	0.998746	0.994395
FNR	0.008282	0.001481	0.06422
Accuracy	0.995	0.9987	0.988012
F-Measure	0.994808	0.996796	0.944444
AUC	0.994892	0.998632	0.965087

TABLE VI. SENSITIVITY ANALYSIS RESULTS

Parameter Set	Learning Rate	Epochs	NSL-KDD	KDD CUP 99	Koyoto 2022
P1	0.1	50	0.94	0.937	0.913
P2	0.3	50	0.97	0.959	0.944
P3	0.5	50	0.995	0.999	0.979
P4	0.7	50	0.991	0.999	0.973
P5	1	50	0.89	0.926	0.851
P6	0.5	10	0.72	0.69	0.583
P7	0.5	25	0.79	0.829	0.811
P8	0.5	70	0.995	0.999	0.979
P9	0.5	100	0.995	0.999	0.979
P10	0.5	200	0.995	0.999	0.979
P11	0.3	200	0.995	0.999	0.978

This is because an increased learning rate results in big steps, and as a consequence, the model has a tendency to miss the ideal convergence point. Epochs are changed while the learning rate remains the same in parameter sets P6 to P10. It is clear from the reduced epochs (P6 and P7) that the model is not being given a long enough period of time to converge. As a result, the highest possible degree of precision is not achieved. The highest levels of precision can be attained when the period is advanced to 50 (P3) and beyond (P8 to P10). It has been noticed that using 50 epochs provides the highest level of accuracy. After reaching this point, increasing the number of epochs will have no effect on the performance because convergence will have already been reached by that point.

It is possible to summarise that the rate of learning plays an essential part in the achievement of successful results. It is vital to locate the best convergence level and the sweet spot that corresponds to it. Any number less than this point necessitates additional time for the model to converge, and any value greater than this point will cause the model to miss the point at which it converges. Epochs represent the number of times that the training data should be iterated through by the model in order to reach convergence. If there are fewer epochs than necessary, the model will not have enough time to converge, and if there are more epochs than necessary, there will be an additional time overhead with no improvement in performance. In addition to that, it will also result in overfitting, which is why it ought to be avoided. Comparisons are made between the HBM model proposed in Part 3 and the APID model proposed in Part 4 in terms of TPR, TNR, Precision, F-Measure, and AUC on the NSL-KDD, KDD CUP 99, and Koyoto 2022+ datasets, which are depicted in Fig. 11 to 19.

A comparison of TPR, TNR, Precision, F-Measure, and AUC on NSL-KDD data demonstrates that the MLDN model exhibits better prediction levels when compared to APID and HBM (Fig. 11 to 13). This is shown by the fact that the MLDN model has a higher AUC.

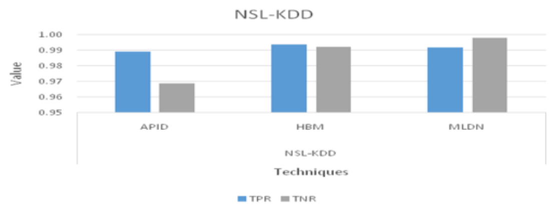

Fig. 11. Comparison of TPR and TNR of MLDN, HBM and APID on NSL-KDD.

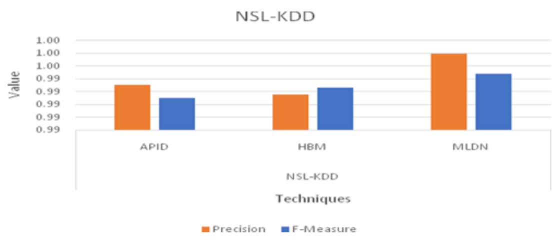

Fig. 12. Comparison of precision and F-Measure of MLDN, HBM and APID on NSL-KDD.

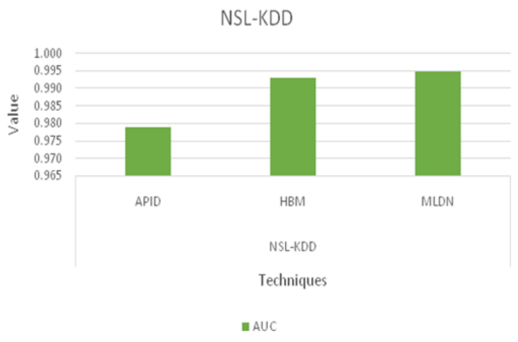

Fig. 13. Comparison of AUC of MLDN of MLDN, HBM and APID on NSL-KDD.

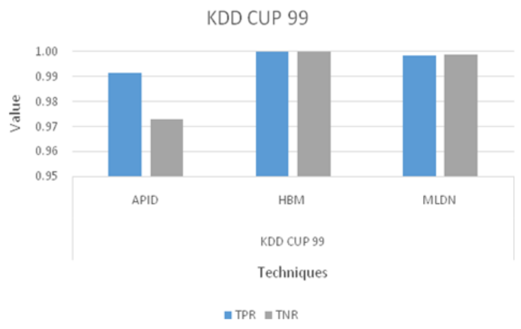

Fig. 14. Comparison of TPR and TNR of MLDN on KDD CUP 99.

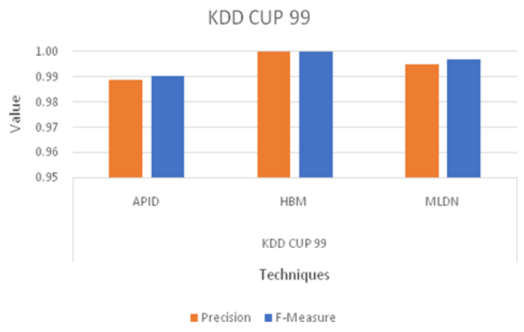

Fig. 15. Comparison of precision and F-Measure of MLDN on KDD CUP 99.

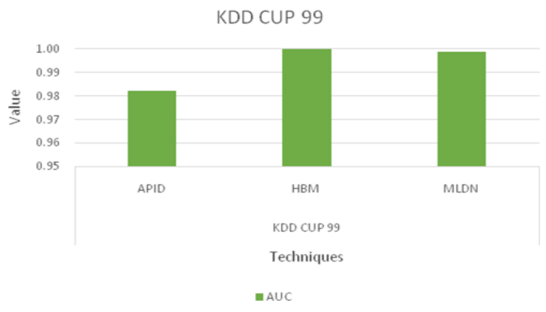

Fig. 16. Comparison of AUC of MLDN on KDD CUP 99.

Analysis of performance on KDD CUP 99 dataset shown in Fig. 14 to 16 demonstrate that in comparison to the APID model, the performance of the MLDN model that has been proposed is superior. In contrast to this, the performance levels demonstrate a marginal drop of 0.1% when measured against the HBM model. The levels of reduction are so negligibly very low that they can be ignored as a result.

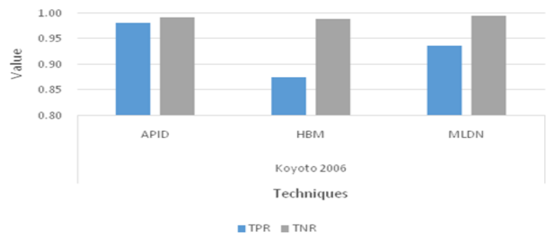

Fig. 17. Comparison of TPR and TNR of MLDN on Koyoto 2022+.

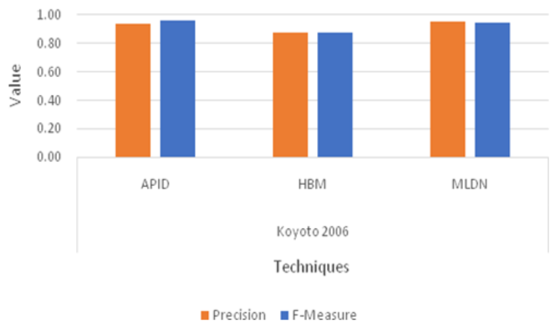

Fig. 18. Comparison of precision and F-Measure of MLDN on Koyoto 2022+.

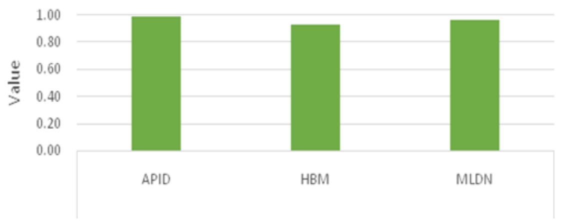

Fig. 19. Comparison of AUC of MLDN on Koyoto 2022+.

When compared to the other models, the performance of the proposed MLDN model on the Koyoto 2022+ datasets shows a slight decrease in the performance with respect to certain metrics such as TPR and F-Measure. Even in this case, the reductions are extremely minimal and, as a result, are insignificant.

Table VII provides a tabular representation of the performance comparisons that were made. The table demonstrates that the overall performance of the proposed models was found to be high and effective, despite the fact that there are slight reductions and elevations in the performance levels of the proposed models.

Table VIII presents a comparison of the amount of time spent training and testing the APID, HBM, and MLDN models. Training is carried out using 70 percent of the records contained in the data, while testing has been carried out using 30 percent of the records across all of the datasets. The HBM model has the most efficiency with regard to its use of time, followed by the APID model and then the MLDN model. Testing requirements for the MLDN model are always less than one second, which is a low requirement that corresponds to a real-time prediction scenario. The training requirements for the MLDN model are quite high. In addition, the little increase in the amount of time needed could be neglected due to the significant boost in terms of performance, which would make the MLDN model the most effective performer when it comes to the detection of intrusions in networks.

The results of a comparative analysis of the proposed MLDN model with the HBM model, the APID model, and models proposed by [5, 6], [7], [8], and [9] are shown in Fig. 20 to 22. This analysis compares the proposed MLDN model with the HBM model and the APID model.

In comparison to other models that are currently available in the literature, the suggested models have a higher level of accuracy in their prediction, as shown in Fig. 20 to 22. The MLDN model produces the best results in terms of performance, followed by the HBM model, which delivers the results with the next best performance. After this comes the APID model, and after that comes the models that already exist in the literature.

TABLE VII. PERFORMANCE COMPARISON OF APID, HBM AND MLDN

Measures	***NSL-KDD***			***KDD CUP 99***			***KOYOTO 2022+***		
	APID	*HBM*	*MLDN*	*APID*	*HBM*	*MLDN*	*APID*	*HBM*	*MLDN*
TPR	*0.99*	*0.99*	*0.99*	*0.99*	*1.00*	*1.00*	*0.98*	*0.88*	*0.94*
Precision	*0.99*	*0.99*	*1.00*	*0.99*	*1.00*	*1.00*	*0.94*	*0.88*	*0.95*
TNR	*0.97*	*0.99*	*1.00*	*0.97*	*1.00*	*1.00*	*0.99*	*0.99*	*0.99*
Accuracy	*0.99*	*0.99*	*1.00*	*0.99*	*1.00*	*1.00*	*0.99*	*0.98*	*0.99*
F-Measure	*0.99*	*0.99*	*0.99*	*0.99*	*1.00*	*1.00*	*0.96*	*0.88*	*0.94*
AUC	*0.98*	*0.99*	*0.99*	*0.98*	*1.00*	*1.00*	*0.99*	*0.93*	*0.97*

TABLE VIII. TIME COMPARISON OF APID, HBM AND MLDN

	NSL-KDD		**KDD CUP 99**		**Koyoto 2022**	
	Training (sec)	**Testing (sec)**	**Training (sec)**	**Testing (sec)**	**Training (sec)**	**Testing(sec)**
MLDN	50	0.259	4	0.316	2	0.279
HBM	2.24	0.21	0.209	0.102	0.107	0.098
APID	2.42	0.302	0.257	0.108	0.0962	0.054

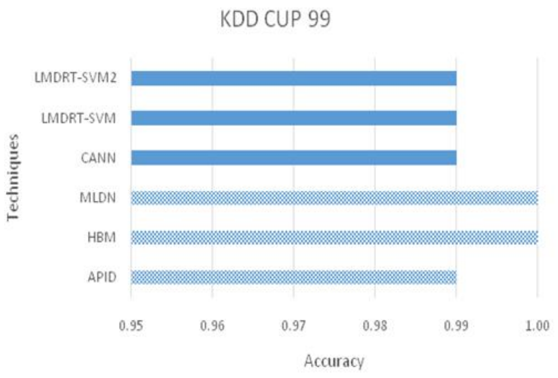

Fig. 20. Comparison of accuracy of MLDN with state-of the-art models on KDD CUP 99.

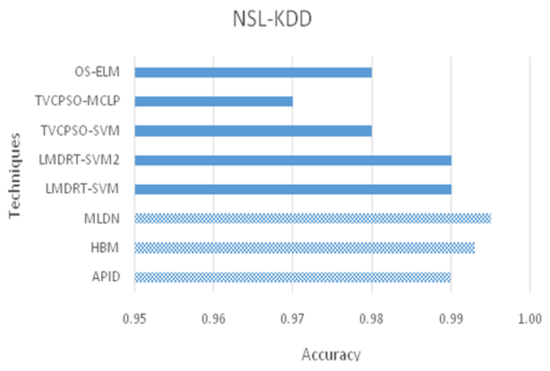

Fig. 21. Comparison of accuracy of MLDN with state-of the-art models on NSL-KDD.

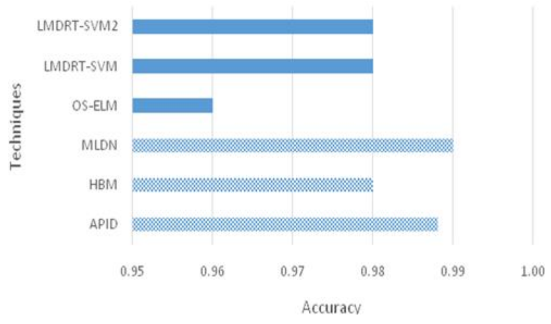

Fig. 22. Comparison of accuracy of MLDN with state-of the-art models on Koyoto 2022+.

A tabular representation of the findings is presented in Table IX. The best results are highlighted in bold below. It was found that the proposed models performed better than all of the existing models that were researched and used as a point of comparison in the literature. In general, methods are constructed with data as their foundation. When utilised with a wider variety of data, such data-specific models are unable to generate results that are beneficial. This particular illustration could be effectively noticed in models from the literature, where models perform better in certain cases while performing worse in others. Because the models that are provided are generic, it is possible to see that the proposed model performs well regardless of the dataset that is being used. This is because the offered models are general.

TABLE IX. PERFORMANCE COMPARISON WITH STATE-OF-THE-ART MODELS

KDD	
Technique	**Accuracy**
APID	0.99
HBM	1.00
MLDN	1.00
CANN	0.99
LMDRT-SVM	0.99
LMDRT-SVM2	0.99
NSL-KDD	
Technique	**Accuracy**
APID	0.99
HBM	0.99
MLDN	1.00
LMDRT-SVM	0.99
LMDRT-SVM2	0.99
TVCPSO-SVM	0.98
TVCPSO-MCLP	0.97
OS-ELM	0.98
Koyoto 2022+	
Technique	**Accuracy**
APID	0.99
HBM	0.98
MLDN	0.99
OS-ELM	0.96
LMDRT-SVM	0.98
LMDRT-SVM2	0.98

V. CONCLUSION

The initial work provides the Adaptive Parallelized Intrusion Detection (APID) model, which is utilised for finding intrusion signs from data that is sent within a network. This model was developed by the researchers. The transmission data that is later sent to the model is first subjected to pre-processing, and then training bags are produced. The training data bags are then given to the learners at the basic level. The training for every base learner is determined by the training bag that is given to it. The learner that demonstrates the best prediction rates in terms of normal data prediction and in terms of the best overall prediction is determined to be the basic learner.

The final predictions are obtained by utilising heuristic-based combiners, after which the test data are forecasted by using all of the basic learners. After that, the ensemble is retrained based on the false predictions to produce an adaptive model that is capable of changing itself to produce better predictions over the course of time. Despite the fact that the model delivers impressive results, it is not appropriate for use for processing large amounts of data because of the significant computational cost connected with it. In the following contribution, a Heterogeneous Bagging based Model, or HBM, is presented with the goal of reducing complexity levels. This model features an improved bagging method that makes it possible to detect intrusions both more quickly and more accurately. The training data is divided up into various bags that also overlap with one another. Both the Decision Tree and Random Forest models are utilised here as the foundation learners for the model. The bagging procedure is altered in such a way that each data bag is made available to both models. Each of the bags that are constructed gets its own unique set of several pairs of base learners.

On the basis of these models, predictions are made, and the results of those predictions are combined with the votes from the voting combiner. While this model does a better job of simplifying complex processes, its performance is marginally inferior. A deep learning network-based intrusion detection model, known as the MLDN, is presented as the last contribution. This is done in order to improve efficiency. The Keras library was utilised during the construction of the deep learning model. In order to accommodate the specific processing architecture of neural networks, the training data is preprocessed and partitioned.

The network is built with various layers, and all of the other necessary parameters for the network are configured based on the data that is entered. Validation of the trained model is accomplished by employing the validation data that has been meticulously isolated for the sole purpose of fulfilling this requirement. Standard benchmark datasets, such as KDD CUP99, NSL-KDD, and Koyoto 2022+ datasets, were utilised in the experiments that were carried out. Comparisons were made with previously published models that were already in existence. The analysis was carried out using the existing standard performance metrics for classifiers, which included TPR, FPR, TNR, FNR, Precision, Recall, F-Measure, and Accuracy. According to the findings, the proposed models appear to have superior performance levels when compared to the standard models that are currently in use.

REFERENCES

[1] B. Gao, B. Bu, W. Zhang and X. Li, "An Intrusion Detection Method Based on Machine Learning and State Observer for Train-Ground Communication Systems," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 6608-6620, July 2022.

[2] R. Bitton and A. Shabtai, "A Machine Learning-Based Intrusion Detection System for Securing Remote Desktop Connections to Electronic Flight Bag Servers," in IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 3, pp. 1164-1181, 1 May-June 2021.

[3] Sahu et al., "Multi-Source Multi-Domain Data Fusion for Cyberattack Detection in Power Systems," in IEEE Access, vol. 9, pp. 119118-119138, 2021.

[4] X. Y. Li, R. Tang and W. Song, "Intrusion Detection System Using Improved Convolution Neural Network," 2022 11th International Conference of Information and Communication Technology (ICTech)), Wuhan, China, 2022, pp. 97-100.

[5] C. Chen, X. Xu, G. Wang and L. Yang, "Network intrusion detection model based on neural network feature extraction and PSO-SVM," 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 2022, pp. 1462-1465.

[6] S. L. Rocha, G. Daniel Amvame Nze and F. L. Lopes de Mendonça, "Intrusion Detection in Container Orchestration Clusters : A framework proposal based on real-time system call analysis with machine learning for anomaly detection," 2022 17th Iberian Conference on Information Systems and Technologies (CISTI), Madrid, Spain, 2022, pp. 1-4.

[7] R. Zhang, Y. Song and X. Wang, "Network Intrusion Detection Scheme Based on IPSO-SVM Algorithm," 2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC), Dalian, China, 2022, pp. 1011-1014.

[8] H. Yang, Y. Bai, T. Chen, Y. Shi, R. Yang and H. Ma, "Intrusion Detection Model For Power Information Network Based On Multi-layer Attention Mechanism," 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2022, pp. 825-828.

[9] F. J. Mora-Gimeno, H. Mora-Mora, B. Volckaert and A. Atrey, "Intrusion Detection System Based on Integrated System Calls Graph and Neural Networks," in IEEE Access, vol. 9, pp. 9822-9833, 2021.

[10] X. Gong, X. Chen, Z. Zhong and W. Chen, "Enhanced Few-Shot Learning for Intrusion Detection in Railway Video Surveillance," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 8, pp. 11301-11313, Aug. 2022.

[11] P. Freitas De Araujo-Filho, A. J. Pinheiro, G. Kaddoum, D. R. Campelo and F. L. Soares, "An Efficient Intrusion Prevention System for CAN: Hindering Cyber-Attacks With a Low-Cost Platform," in IEEE Access, vol. 9, pp. 166855-166869, 2021.

[12] M. Abdel-Basset, N. Moustafa, H. Hawash, I. Razzak, K. M. Sallam and O. M. Elkomy, "Federated Intrusion Detection in Blockchain-Based Smart Transportation Systems," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 3, pp. 2523-2537, March 2022.

[13] Q. Liu, V. Hagenmeyer and H. B. Keller, "A Review of Rule Learning-Based Intrusion Detection Systems and Their Prospects in Smart Grids," in IEEE Access, vol. 9, pp. 57542-57564, 2021.

[14] C. Jichici, B. Groza, R. Ragobete, P. -S. Murvay and T. Andreica, "Effective Intrusion Detection and Prevention for the Commercial Vehicle SAE J1939 CAN Bus," in IEEE Transactions on Intelligent Transportation Systems. doi: 10.1109/TITS.2022.3151712.

[15] J. Shu, L. Zhou, W. Zhang, X. Du and M. Guizani, "Collaborative Intrusion Detection for VANETs: A Deep Learning-Based Distributed SDN Approach," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4519-4530, July 2021.

[16] M. Nadeem, A. Arshad, S. Riaz, S. S. Band and A. Mosavi, "Intercept the Cloud Network From Brute Force and DDoS Attacks via Intrusion Detection and Prevention System," in IEEE Access, vol. 9, pp. 152300-152309, 2021.

[17] T. Wisanwanichthan and M. Thammawichai, "A Double-Layered Hybrid Approach for Network Intrusion Detection System Using Combined Naive Bayes and SVM," in IEEE Access, vol. 9, pp. 138432-138450, 2021.

[18] G. Apruzzese, L. Pajola and M. Conti, "The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems," in IEEE Transactions on Network and Service Management.

[19] J. Lansky et al., "Deep Learning-Based Intrusion Detection Systems: A Systematic Review," in IEEE Access, vol. 9, pp. 101574-101599, 2021.

[20] L. Yang, A. Moubayed and A. Shami, "MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles," in IEEE Internet of Things Journal, vol. 9, no. 1, pp. 616-632, 1 Jan.1, 2022.

[21] Y. Miao, Y. Tang, B. A. Alzahrani, A. Barnawi, T. Alafif and L. Hu, "Airborne LiDAR Assisted Obstacle Recognition and Intrusion Detection Towards Unmanned Aerial Vehicle: Architecture, Modeling and Evaluation," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4531-4540, July 2021.

[22] M. A. Siddiqi and W. Pak, "An Agile Approach to Identify Single and Hybrid Normalization for Enhancing Machine Learning-Based Network Intrusion Detection," in IEEE Access, vol. 9, pp. 137494-137513, 2021.

[23] Y. Gao, H. Miao, J. Chen, B. Song, X. Hu and W. Wang, "Explosive Cyber Security Threats During COVID-19 Pandemic and a Novel Tree-Based Broad Learning System to Overcome," in IEEE Transactions on Intelligent Transportation Systems.

[24] Z. Hu, S. Liu, W. Luo and L. Wu, "Intrusion-Detector-Dependent Distributed Economic Model Predictive Control for Load Frequency Regulation With PEVs Under Cyber Attacks," in IEEE Transactions on

Circuits and Systems I: Regular Papers, vol. 68, no. 9, pp. 3857-3868, Sept. 2021.

[25] R. Conde Camillo da Silva, M. P. Oliveira Camargo, M. Sanches Quessada, A. Claiton Lopes, J. Diassala Monteiro Ernesto and K. A. Pontara da Costa, "An Intrusion Detection System for Web-Based Attacks Using IBM Watson," in IEEE Latin America Transactions, vol. 20, no. 2, pp. 191-197, Feb. 2022.

[26] K. Agrawal, T. Alladi, A. Agrawal, V. Chamola and A. Benslimane, "NovelADS: A Novel Anomaly Detection System for Intra-Vehicular Networks," in IEEE Transactions on Intelligent Transportation Systems.doi: 10.1109/TITS.2022.3146024.

[27] C. Kim, M. Jang, S. Seo, K. Park and P. Kang, "Intrusion Detection Based on Sequential Information Preserving Log Embedding Methods and Anomaly Detection Algorithms," in IEEE Access, vol. 9, pp. 58088-58101, 2021.

[28] J. Gao et al., "Omni SCADA Intrusion Detection Using Deep Learning Algorithms," in IEEE Internet of Things Journal, vol. 8, no. 2, pp. 951-961, 15 Jan.15, 2021.

[29] T. Kim and W. Pak, "Hybrid Classification for High-Speed and High-Accuracy Network Intrusion Detection System," in IEEE Access, vol. 9, pp. 83806-83817, 2021.

[30] N. Mishra and S. Pandya, "Internet of Things Applications, Security Challenges, Attacks, Intrusion Detection, and Future Visions: A Systematic Review," in IEEE Access, vol. 9, pp. 59353-59377, 2021.

[31] G. Pu, L. Wang, J. Shen and F. Dong, "A hybrid unsupervised clustering-based anomaly detection method," in Tsinghua Science and Technology, vol. 26, no. 2, pp. 146-153, April 2021.

[32] O. Alkadi, N. Moustafa, B. Turnbull and K. -K. R. Choo, "A Deep Blockchain Framework-Enabled Collaborative Intrusion Detection for Protecting IoT and Cloud Networks," in IEEE Internet of Things Journal, vol. 8, no. 12, pp. 9463-9472, 15 June15, 2021.

[33] S. Seth, K. K. Chahal and G. Singh, "A Novel Ensemble Framework for an Intelligent Intrusion Detection System," in IEEE Access, vol. 9, pp. 138451-138467, 2021.

[34] D. Gümüşbaş, T. Yıldırım, A. Genovese and F. Scotti, "A Comprehensive Survey of Databases and Deep Learning Methods for Cybersecurity and Intrusion Detection Systems," in IEEE Systems Journal, vol. 15, no. 2, pp. 1717-1731, June 2021.

[35] H. Janabi, T. Kanakis and M. Johnson, "Overhead Reduction Technique for Software-Defined Network Based Intrusion Detection Systems," in IEEE Access, vol. 10, pp. 66481-66491, 2022.

[36] L. Vu, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang and E. Dutkiewicz, "Deep Generative Learning Models for Cloud Intrusion Detection Systems," in IEEE Transactions on Cybernetics.doi: 10.1109/TCYB.2022.3163811.