

# DuRBIN: A Comprehensive Approach to Analysis and Detection of Emerging Threats due to Network Intrusion

Kumar Priyansh<sup>1,\*</sup>, Ritu Dimri<sup>1</sup>, Fahim Islam Anik<sup>2</sup>, Hossain Shahriar<sup>3</sup>, Zakirul Alam Bhuiyan<sup>4</sup>, and Nazmus Sakib<sup>3,\*</sup>

<sup>1</sup>Institute for Artificial Intelligence and Data Science, University at Buffalo, The State University of New York, New York, United States

<sup>2</sup>Department of Mechanical Engineering, Khulna University of Engineering & Technology (KUET), Khulna, Bangladesh

<sup>3</sup>Department of Information Technology, Kennesaw State University, Marietta, Georgia, United States

<sup>4</sup>Department of Computer and Information Sciences, Fordham University, New York, United States

\*Corresponding authors: [kumarpri@buffalo.edu](mailto:kumarpri@buffalo.edu), [nsakib1@kennesaw.edu](mailto:nsakib1@kennesaw.edu)

**Abstract**— Network Intrusion Detection System (NIDS) is a popular software application that scans and monitors for suspicious network traffic and alerts when a potential threat is identified. Although NIDS has its' advantages, there are scenarios pertaining to a busy or large network, encrypted data information, and fake addresses that result in frequent false positives and other shortcomings. It is imperative to understand and compare different machine learning algorithms used in NIDS to address these shortcomings and find a better approach for analysis and detection. This research contributes specifically focused on finding that better approach by comparing different machine learning algorithms, such as Deep Learning, Distributed Random Forest, Gradient Boosting Machine, and Naïve Bayes, and demonstrate the eligibility, applicability, and appropriateness of each algorithm with numerical values and graphs. 'LUFlow' Network Intrusion Detection Data Set was used to conduct the research. Additionally, to aid in visualizing the algorithm outcomes based on malicious networks, benign networks, and outliers, a 'RShiny' interface is developed where users can see different plots for different variables of concern, check predictions, track model metrics, and have a model summary. The derived results of this research can help in the methodological approach of implementing various machine learning algorithms in NIDS while addressing various existing limitations.

**Keywords**— NIDS, Machine Learning, LUFlow, Deep Learning, Network Intrusion Detection

## I. INTRODUCTION

In this era of web-based technological advancements and the Internet of things (IoT), the number of interconnected network systems among various domains related to business, communication, and collaboration in various organizations and industries is increasing continuously [1], [2]. Modern interconnected business environments need a high level of security to maintain safe and trusted communication of information between different organizations [3]. Recently, IC3 saw a 69% increase in network-related complaints from 2019 as per the 2020 FBI Internet crime report. With the continuous network-based communication systems, cyber-attacks are most likely to become more sophisticated, and hence it is important that protection technologies adapt along with their threats [4]. Therefore, security, integrity, availability, and confidentiality in different networks become a major area of concern, especially after attacks such as VPNFilter [5] and Dyn [6], [7].

In such a circumstance, an Intrusion Detection System works as a highly useful tool for organizations of all sizes that regularly deal with different kinds of network-based attacks on their servers. Such a system can identify and block

malicious connections before they can even reach the main servers. All kinds of organizations ranging from educational to governmental, can benefit from such a system. It works identical to an adaptable security-based safeguard technology after traditional technologies fail [8].

Researchers are focusing on different intrusion detection tools and methods to prevent malicious attacks [9]. However, there is a wide range of problems pertaining to NIDS, such as encrypted messages, packaged data, fake addresses, large networks, etc. Hence, different approaches in terms of applying machine learning algorithms for specific applications need to be tested to address these issues. Many approaches to solutions have been discussed that talk about various practical methods [10], [11]. Design, comparison, and evaluation of different algorithms and approaches are sectors that are being focused on [12]–[14].

'DuRBIN' focuses on testing different machine learning algorithms such as naïve bayes, deep learning, gradient boosting machine, and random forest in NIDS and shows comparative performance analysis. Although, Network Intrusion Detection Systems are highly efficient devices used to monitor network activity, such devices are prone to a lot of errors and shortcomings like identifying false positives, fake IP packets, inability to detect encrypted packets, among many others. 'DuRBIN' will help future researchers to choose which methodical machine learning approach to take based on their specific application and purpose. Keeping that aim in mind, the present research was conducted using the 'LUFlow' network intrusion and detection data set. By looking at the different features and behaviors of the variables present in Data Set, the aim of this research is to build four different machine learning models and a 'RShiny' application to compare the performance of the machine learning models. The interface to visually analyze, compare and understand output data was developed using 'RShiny' where a user can navigate all the details similar to a mobile application containing different tabs. The unique contributions of this article to the body of knowledge include:

- Comparison of different machine learning technique-based approaches in NIDS.
- Development of a visual tool to analyze different parameters of performances in NIDS.
- Identifying the applicability of different machine learning techniques such as deep learning, random forest, gradient boosting machine, and naïve bayes

with confusion matrixes, importance values, learning curves, etc.

Throughout the work, a comprehensive approach to analysis and detection of emerging threats due to network intrusion is introduced with relevant background analysis. Maintaining coherence to that, different machine learning techniques for NIDS performance evaluation with a 'RShiny' application is proposed.

The rest of the paper has the following organization. Section II introduces a step-by-step research workflow based on data set descriptions, data preprocessing, exploratory data analysis and feature selection, developing and evaluating machine learning models, and 'RShiny' implementation. Following that, section III shows both numerical and graphical comparisons for four different machine learning techniques. Additionally, illustrative examples of the 'RShiny' application based on five different parameters are discussed. Finally, section IV contains a summary of the work with future work recommendations.

## II. APPROACH

Throughout this section, the sequential steps of the research have been explained that includes, data set descriptions, data preprocessing, exploratory data analysis and feature selection, development and evaluation of different machine learning models and "RShiny" implementation as sketched in figure 1.

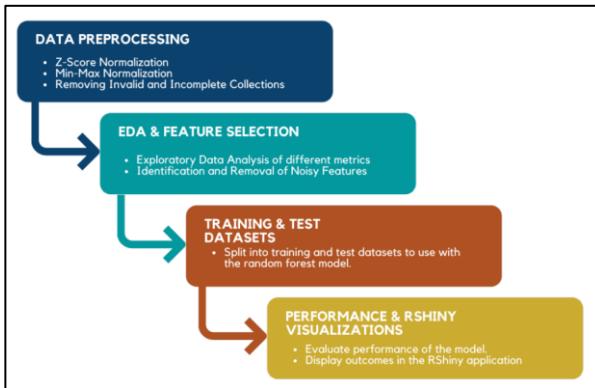


Fig. 1. Flowchart representation of the research workflow.

### A. Data Set Descriptions

'LUFlow Network Intrusion Detection Data Set' [15] was used for this research. Out of 14 feature variables that the dataset contains, six features were selected that were highly correlated and were of the highest importance for the research. The feature selection methods were applied using the caret package. These variables are:

- num\_pkts\_in: The packet-count from source to destination
- bytes\_in: The number of bytes transmitted from source to destination
- num\_pkts\_out: The packet-count from destination to source
- bytes\_out: The number of bytes transmitted from destination to source
- dest\_port: The destination port number associated with the flow

- total\_entropy: The total entropy in bytes over all of the bytes in the data fields of the flow

The metadata [16] of this research contains all the necessary information to understand each step and evaluate accordingly.

### B. Data Preprocessing

Initially, data processing was done using Z-score normalization of the data. The widely known standardization method is Z-score standardization, also known as standard deviation standardization and is defined as:

$$Y_{zscore} = \frac{Y_{inst} - U}{\delta} \quad (1)$$

In equation (1),  $Y_{inst}$  is the value of initialized feature,  $U$  denotes the mean feature vector, and  $\delta$  is the standard deviation. Then min-max normalization is carried out.

$$\psi_{ij} = \frac{Y_{ij} - \min(Y_j)}{\max(Y_j) - \min(Y_j)} * N \quad (2)$$

In equation (2),  $\psi_{ij}$  denotes the normalized value of  $Y_{ij}$  with a range of 0 to  $N$  in the form of integer,  $\min(Y_j)$  indicates the least possible value of the  $j^{\text{th}}$  feature, and  $\max(Y_j)$  is the highest possible value of the  $j^{\text{th}}$  feature.

The research was carried forward with the removal of rows with missing and incomplete data to accomplish data preprocessing and move to the next steps.

### C. Exploratory Data Analysis and Feature Selection

Identifying noisy features is necessary as they are not directly related to cyberattacks and have no causal effect in terms of classified outcomes.

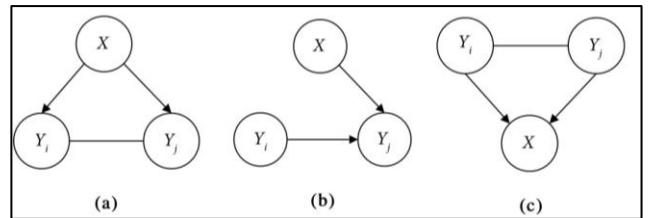


Fig. 2. Simplified Illustration of the Influences of Features on Cyber-attacks [17].

Figure 2 demonstrates the different relations and connections among cyberattack  $X$  and feature  $Y$  for general cases. Identifying an accurate type of cyberattack is inhibited if the direction and causal relationship among these parameters are not clarified properly. In Figure 2(b), the assumption was that  $Y_i$  and  $Y_j$  are in a mutually causal relationship. Therefore, any anomaly in one of the features will have a similar effect on the other one. Hence, if the anomalous feature  $Y_j$  is considered to be caused by the cyberattack  $X$ , there is a possibility of a wrong conclusion. Figure 2(c) illustrates the reversal of the causal direction between cyberattack  $X$  and feature  $Y$ . Therefore, with the interference of feature  $Y$ , the causal relationship between  $Y$  and  $X$  can be worked out according to changes in the expected value of  $X$ , which can be formulated as:

$$E[X|do(Y_i)] = \sum_x xP(x|do(Y_i = y_i)) \quad (3)$$

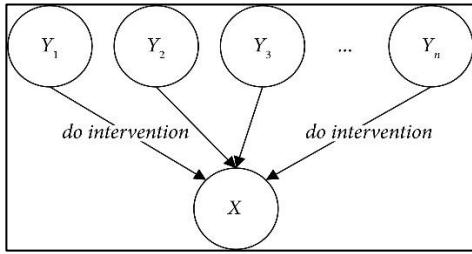


Fig. 3. Intervention Process [17].

If there is no causal relationship between  $X$  with  $Y_1, Y_3, \dots, Y_{n-1}$  and other features, as shown in Figure 3, then it can be written:

$$E[X|do(Y_i), do(Y_3), do(Y_{n-1})] = \sum_x \sum_x \dots \dots \sum_x xP(x|y_2, \dots, y_n) P(y_n) \quad (4)$$

With the assumption that equation 4 holds, the causal relationship in the case can be recovered based on the factual causal direction between anomalous features and cyberattacks, as illustrated in Figure 4.

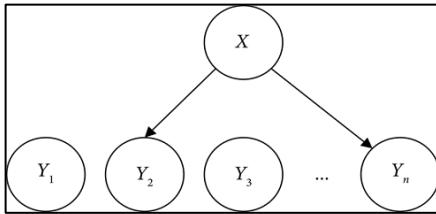


Fig. 4. A Factual Causal Relationship between a Single Cyberattack and Features [17].

If intervention is made on  $Y_1, Y_3, \dots, Y_{n-1}$ , then the intensity of causal effect between  $X_k$  and  $Y_1, Y_3, \dots, Y_{n-1}$  is:

$$\xi_k = \sum_x |E[Y_i = 1 | do(X_k = 1)] - E[Y_i = 1 | do(X_k = 0)]| \quad (5)$$

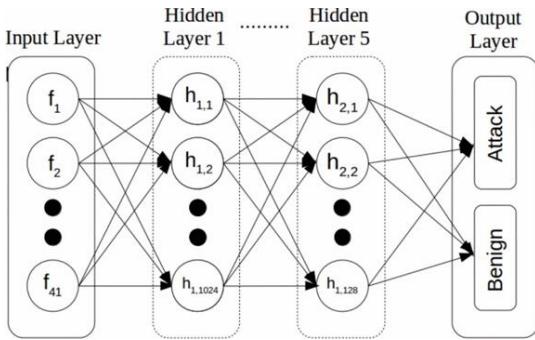


Fig. 5. Proposed Architecture of the DNN.

Based on discussions above, Figure 5 shows the proposed architecture of the DNN including input, hidden and output layers.

#### 1) Input and hidden layers

Initially the number of layers sum to a total of 41 neurons. Then these are fed into the hidden layers. The non-linear activation function, ‘ReLU’ [18], [19] is used for hidden layers. After that, weights are added to feed them forward to the next hidden layer. For the purpose of accurate outputs along with the reduction of computational time and cost, the neuron count in each hidden layer is decreased steadily.

#### 2) Regularization

To make the whole process efficient and time-saving, Dropout[20], [21] is used. Dropout is used to unplug the neurons randomly to make the model robust and prevent from overfitting the training set.

#### 3) Output layer and classification

A sigmoid activation function [22] is used to convert the 1024 neurons from the previous layer into only two neurons. Hence, the output layer has two neurons, Attack and Benign. The sigmoid function, due to its nature returns just two outputs favoring the binary classification intended for this research.

#### D. Developing and Evaluating Machine Learning Models

Four machine learning models were implemented and investigated to identify and evaluate the effects of different parameters pertaining to NIDS.

- Deep Learning: The first implemented model was deep learning. A multi-layer feedforward artificial neural network was used. It was trained with stochastic gradient descent using backpropagation. Performance analysis based on malicious attack detection, outlier, and benign network identification was done.
- Distributed Random Forest: Distributed random forest (DRF) can work as a tool for classifying based on majority vote and probability to make smart predictions and identifications. Instead of a single regression or classification tree decision, DRF works with a forest of classifications for better output.
- Gradient Boosting Machine: The gradient boosting machine converts weak learners into strong ones through iterative operations. Usually, it outperforms the random forest model. This method was applied to the data set, and performance output was compared through a confusion matrix and other indices.
- Naïve Bayes: Naïve Bayes can also be implemented for the purpose of network intrusion detection. It is trained with a large amount of data set as input. The greater the data set, the better the result. It makes a decision based on the highest probability for a given problem.

Section III further describes the comparison and interpretation of each model in detail.

#### E. ‘RShiny’ Implementation

‘RShiny’ [23] application primarily has two key components, a server that runs R code and a User Interface (UI) that runs through a user’s web browser. The UI has input and output fields. Input fields takes data and sends it to the server where processing occurs and it gets back to the UI to be displayed.

The interactive interface was created from the point of view of a general user so that, any individual can input data easily, visualize them in output in forms of different graphs, make predictions and evaluate performance based on model metrics and model summary.

### III. RESULTS AND DISCUSSIONS

This section consists of two core parts consisting of (a) comparison of performance and interpretation of the four machine learning models based on the methodology as discussed in section II, and (b) utility with illustrative examples of the developed ‘RShiny’ app.

#### A. Comparing and Interpreting Models

##### 1) Deep Learning Model:

The deep learning model for this research was built using H2O’s Deep Learning model [24] , which is a multi-layer feedforward artificial neural network that is trained with stochastic gradient descent using backpropagation. The confusion matrix was found is shown in Table I.

It can be clearly observed in Table I that malicious (1) and benign networks (2) were predicted to a satisfactory level, but the outliers (3) were not predicted as outliers. This is not necessarily a not up to par outcome. In real life scenario, the network, which was labeled as an outlier, must have originally belonged to either of the malicious or benign classes. But, since they were unidentified at that point, they were labeled as outliers. The reason occurrences like these might occur is because, the model tried and segregated the outliers into either or both of the classes as mentioned above.

In the end, it can be seen from Table I, that the final error was found to be 0.1197, giving the model an accuracy of 88.03% – which is a high accuracy considering the difficulty of the problem. The model is a Multinomial Deep Learning model built with three layers (input, tanh, SoftMax). The learning curve can be seen in Figure 6.

The variables which were given the most priority by the model while making a prediction are defined by the variable importance as shown in Table II. There are six variables of concern as previously discussed in section II. Relative importance and scaled importance for the deep learning model with percentage values can be seen in table II.

TABLE I. CONFUSION MATRIX FOR DEEP LEARNING MODEL

Row labels: Actual class; Column labels: Predicted class					
	1	2	3	Error	Rate
1	407752	1391	3	0.0034 =	1,394 / 409,146
2	20243	343166	42	0.0558 =	20,285 / 363,451
3	879	72991	57	0.9992 =	73,780 / 73,927
<b>Totals</b>	428874	417548	102	0.1129 =	95,549 / 846,524

TABLE II. VARIABLE IMPORTANCE FOR DEEP LEARNING MODEL

	variable	relative_importance	scaled_importance	Percentage
1	dest_ip	1.000000	1.000000	0.907645
2	dest_port	0.075506	0.075506	0.068532
3	bytes_out	0.020215	0.020215	0.018348
4	entropy	0.004272	0.004272	0.003877
5	bytes_in	0.001184	0.001184	0.001075
6	avg_ipt	0.000575	0.000575	0.000522

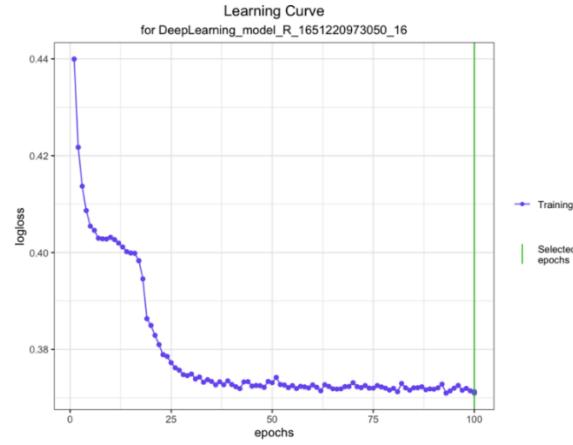


Fig. 6. Learning Curve of the Deep Learning Model.

##### 2) Distributed Random Forest

When given a set of data, Distributed Random Forest (DRF) generates a forest of classification or regression trees rather than a single classification or regression tree. The confusion matrix for this approach is shown in Table III. As we can see in the above output in table III, unlike the Deep Learning Model, the DRF model has generated outlier data. The error is found to be 0.1129, giving the model an accuracy of 88.71% – which is slightly better than the deep learning model. The model is built to have ten trees and a depth of 5. There were 17 minimum leaves and 32 maximum leaves.

By looking at the graph in Figure 7, we can say that the model is more generalizable because the gap between the training and validation curves is less, and they follow the same trend too. Table IV shows the relative and scaled variable importance for random forest model.

TABLE III. CONFUSION MATRIX FOR DISTRIBUTED RANDOM FOREST

Row labels: Actual class; Column labels: Predicted class					
	1	2	3	Error	Rate
1	498481	18512	0	0.0358 =	18,512 / 516,993
2	16335	442309	0	0.0356 =	16,335 / 458,644
3	218	92833	0	1.0000 =	93,051 / 93,051
<b>Totals</b>	515034	553654	0	0.1129 =	127,898 / 1,068,688

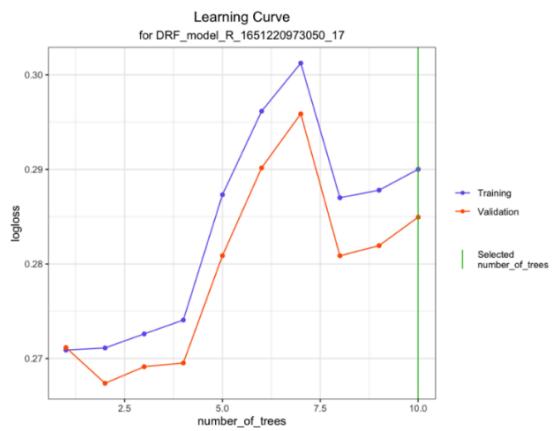


Fig. 7. Learning Curve of the Distributed Random Forest Model

TABLE IV. VARIABLE IMPORTANCE FOR DISTRIBUTED RANDOM FOREST MODEL

	<i>variable</i>	<i>relative_importance</i>	<i>scaled_importance</i>	<i>Percentage</i>
<b>1</b>	dest_port	1311871.625000	1.000000	0.494941
<b>2</b>	bytes_out	925692.562500	0.705627	0.349244
<b>3</b>	total_entropy	279315.875000	0.212914	0.105380
<b>4</b>	bytes_in	68193.351562	0.051982	0.025728
<b>5</b>	num_pkts_out	40154.921875	0.030609	0.015150
<b>6</b>	num_pkts_in	25333.812500	0.019311	0.009558

TABLE V. CONFUSION MATRIX FOR GRADIENT BOOSTING MACHINE

Row labels: Actual class; Column labels: Predicted class					
	<b>1</b>	<b>2</b>	<b>3</b>	<b>Error</b>	<b>Rate</b>
<b>1</b>	516878	115	0	0.0002 =	115 / 516,993
<b>2</b>	282	458161	201	0.0011 =	483 / 458,644
<b>3</b>	2	90347	2702	0.9710 =	90,349 / 93,051
<b>Totals</b>	517162	548623	2903	0.0851 =	90,947 / 1,068,688

### 3) Gradient Boosting Machine

Gradient Boosting Machine (GBM) is a forward learning ensemble method. From the confusion matrix shown in Table V an error of 0.0851 can be seen, which gives us an accuracy of 91.49%. However, it is seen that the GBM Model has described more outputs as outliers (n: 2903) while DRF had 102 and the deep learning model had none. This accuracy might come at the risk of more data being classified as outliers.

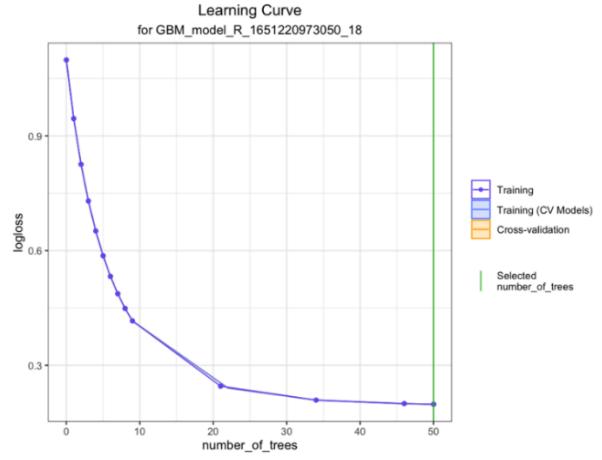


Fig. 8. Learning Curve of the Gradient Boosting Machine Model

## A Comprehensive Approach To Analysis and Detection of Emerging Threats due to Network Intrusion

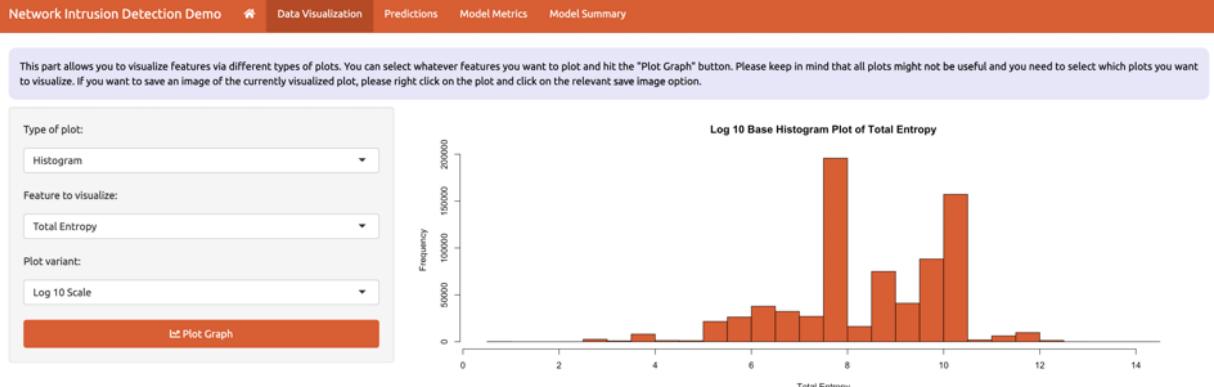


Fig. 9. Data Visualization Screen Showing Histograms on Log-10 Scale

## A Comprehensive Approach To Analysis and Detection of Emerging Threats due to Network Intrusion

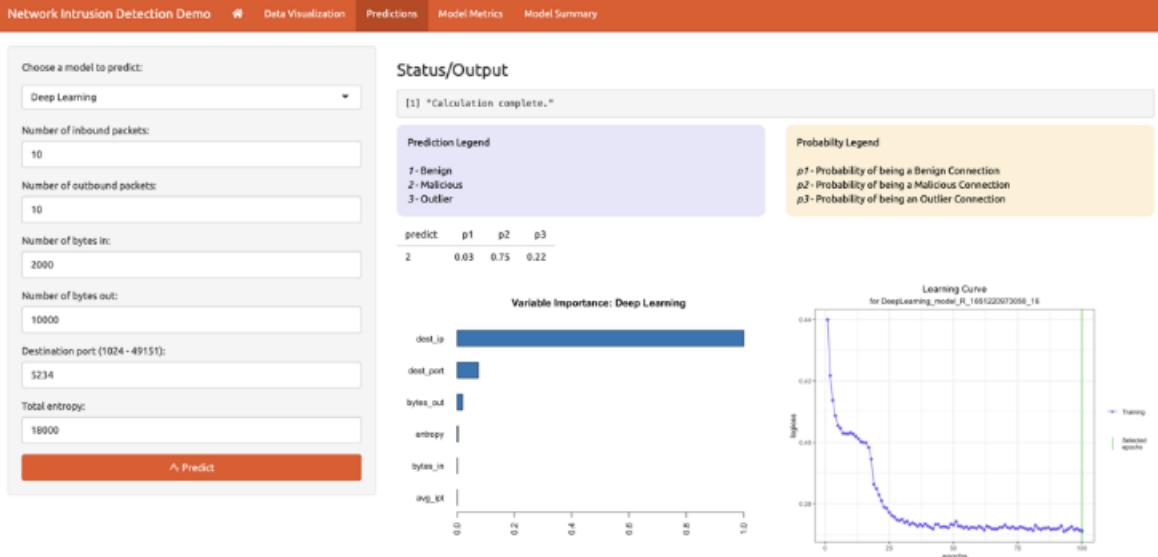


Fig. 10. Predictions Screens running a sample prediction for the Deep Learning Model

TABLE VI. VARIABLE IMPORTANCE FOR GRADIENT BOOSTING MACHINE MODEL

	<i>variable</i>	<i>relative_importance</i>	<i>scaled_importance</i>	<i>Percentage</i>
<b>1</b>	dest_port	2246234.500000	1.000000	0.951435
<b>2</b>	num_pkts_in	52991.281250	0.023591	0.022445
<b>3</b>	num_pkts_out	44265.160156	0.019706	0.018749
<b>4</b>	bytes_out	9491.684570	0.004226	0.004020
<b>5</b>	bytes_in	5477.406738	0.002438	0.002320
<b>6</b>	total_entropy	2431.772461	0.001083	0.001030

TABLE VII. CONFUSION MATRIX FOR NAÏVE BAYES MODEL

Row labels: Actual class; Column labels: Predicted class					
	<b>1</b>	<b>2</b>	<b>3</b>	<b>Error</b>	<b>Rate</b>
<b>1</b>	206354	308867	1772	0.6009 =	310,639 / 516,993
<b>2</b>	7950	446445	4249	0.0266 =	12,199 / 458,644
<b>3</b>	2823	90088	140	0.9985 =	92,911 / 93,051
<b>Totals</b>	217127	845400	6161	0.3890 =	415,749 / 1,068,688

The model has 50 trees and a depth of 5. There are 24 minimum leaves, and 32 is the maximum limit for the same. The learning curve and variable importance are shown in Figure 8 and Table VI.

#### 4) Naïve Bayes Model

The Naïve Bayes model worked as base for the research, to test the Naïve Bayes Classifier. The confusion matrix was

found as shown in Table VII. As seen in the table, the error is 0.3890, giving an accuracy of 61.10% – which is the least accuracy found so far. The classification of the outliers is also found to be huge.

#### B. Utility and Illustrative Examples of 'RShiny' App

To bring all the models together, an interface was built using the 'RShiny' package to show case visualization, metrics, model summaries and to let users predict any network activity as benign, malicious or outlier. There are 5 parts to the application:

##### 1) Home

The Home section contains a table view of the original, unmodified dataset and helps the user to explore the dataset.

##### 2) Data Visualization

It helps to look at the different varieties of plots of different features. The visualizations are run on the cleaned dataset. There are three types of plots provided:

- Histograms
- Scatter Plots
- Mosaic Plot

Histograms are the default type of plots and are available to be visualized on two scales, viz., normal scale, and log-10 scale, as seen in Figure 9.

## A Comprehensive Approach To Analysis and Detection of Emerging Threats due to Network Intrusion

The screenshot shows the RShiny application interface for a Deep Learning model. The top navigation bar includes tabs for 'Network Intrusion Detection Demo', 'Data Visualization', 'Predictions', 'Model Metrics', and 'Model Summary'. The 'Model Summary' tab is currently selected. Below the tabs, there is a dropdown menu labeled 'Choose a model to show metrics:' with 'Deep Learning' selected. A large orange button labeled 'Show Information' is visible. The main content area displays the following information:

**Model Details:**

```
H2OMultinomialModel: deeplearning
Model Key: DeepLearning_model_R_1651228973050_16
Status of Neuron Layers: predicting label, 3-class classification, multinomial distribution, CrossEntropy loss, 13 weights/biases, 3,5 layer units
layer type dropout 11 12 mean_rate_mean_weight_mean_bias
1 1 6 Input 0.00 % NA NA NA NA NA NA NA NA
2 2 1 Tanh 0.00 % 0.000000 0.000000 0.004757 0.005052 0.000000 -18.852732 48.575226 -36.588257
3 3 3 Softmax NA 0.000000 0.000000 0.001915 0.000828 0.000000 248.172114 33.548286 -83.404721
bias_rms
1 NA
2 0.000000
3 33.336426
```

**H2OMultinomialMetrics:**

```
deeplearning
** Reported on training data. **
** Metrics reported on full training frame **
```

**Training Set Metrics:**

```
Extract training frame with 'h2o.getFrame("dataset_sid_bbc0_1")'
MSE: (Extract with 'h2o.mse') 0.1050521
RMSE: (Extract with 'h2o.rmse') 0.3241175
Logloss: (Extract with 'h2o.logloss') 0.3709698
Mean Per-Class Error: 0.357143
AUC1: (Extract with 'h2o.auc') NaN
AUCCP: (Extract with 'h2o.aucpr') NaN
Confusion Matrix: Extract with 'h2o.confusionMatrix(<model>,train = TRUE)'
```

**Confusion Matrix:**

Row labels: Actual class; Column labels: Predicted class				
1	2	3	Error	Rate
1 498481 18512 0 0.0358 = 18,512 / 516,993	2 16335 442309 0 0.0356 = 16,335 / 458,644	3 218 92833 0 1.0000 = 93,051 / 93,051	Totals 515834 553654 0 0.1197 = 127,898 / 1,068,688	

**Hit Ratio Table:**

```
Extract with 'h2o.hit_ratio_table(<model>,train = TRUE)'
```

**Top-3 Hit Ratios:**

k	hit_ratio
1 1	0.888322
2 2	0.987648
3 3	1.000000

Fig. 11. Model Summary with the Deep Learning Model.

### 3) Predictions

The user can input network activity data here and can predict the outcome as shown in Figure 10. The prediction screen contains the input boxes to input the various features required by the algorithm and predicts the outcome. The outcomes are classified into three categories. They are:

- Benign
- Malicious
- Outlier

The prediction Table contains 4 outcomes:

- predict - The class of the prediction, i.e., 1, 2, 3.
- p1 - Probability of being a Benign Connection
- p2 - Probability of being a Malicious Connection
- p3 - Probability of being an Outlier Connection

This screen also contains graphs for the different models. The two types of graphs which are shown in Figure 10 are Variable Importance plot and the Learning Curve plot. Naive Bayes does not support these plots; hence they are not available for the Naive Bayes Model.

### 4) Model Metrics

The model metrics screen shows the metrics for each model. The screen offers a drop-down list of models to choose from and see the metrics. These metrics are extremely helpful when choosing a model to trust on.

### 5) Model Summary

Model Summary screen shows all the details of the models as shown in Figure 11. Again, the screen offers a drop-down list of models to choose from and see the details. This screen is very useful for looking at the intricate details of the models.

## IV. CONCLUSIONS AND FUTURE RESEARCH

NIDS is now an important aspect in all recent ICT based systems for maintaining cyber safety in various interconnected organizations. Integration of machine learning with the contextual understanding of different machine learning approaches for different applications has become a must for reasons such as, removing uncertainty in finding the types of cyber-attacks and malicious behavior and the increased complexity of the entire web-based system.

As discussed in the above sections, this research was able to build different machine learning models with great classification accuracy and showed how a visual application could be useful for individuals in different domains. The developed 'RShiny' application tool will also help anyone who wants to see if a connection is benign or malicious to determine, just by entering some values. Usually, the data which user inputs are stored in a log monitored by the sysadmins and the saved values can be utilized to enter data in the 'RShiny' application and know the class of the connection.

However, this application comes with some limitations too and a great scope for future work in the nearly untouched domain. Right now, the model was built with a subset of the dataset due to limitations of our machines as the complete dataset size sits above 81 gigabytes. A feature which can be

added to this application in the future is "Live Connection Analysis." The idea here is to keep building a "real-time" dataset – which can be any stream of data coming from the logging database and the models are being built in the background while the pre-built models are continuously segregating the new incoming connections and will try to block malicious connections whose predictions are over a threshold (e.g., 98% malicious). It has been noted from the current study that for various attack types majority of the proposed NIDS approaches show a lower detection accuracy than the detection accuracy of the model when viewed holistically. The imbalanced nature of the dataset is behind this issue. The ability to solve the class imbalance problem encountered here could be another future research opportunity.

## REFERENCES

- [1] “Research and Innovation Brief - Annual Report on Cybersecurity Research and Innovation Needs and Priorities — ENISA.” <https://www.enisa.europa.eu/publications/research-and-innovation-brief> (accessed Jun. 14, 2022).
- [2] “Internet Live Stats - Internet Usage & Social Media Statistics.” <https://www.internetlivestats.com/> (accessed Jun. 14, 2022).
- [3] Z. Zeng, W. Peng, and B. Zhao, “Improving the Accuracy of Network Intrusion Detection with Causal Machine Learning,” *Security and Communication Networks*, vol. 2021, 2021, doi: 10.1155/2021/8986243.
- [4] V. K. Rahul, R. Vinayakumar, K. Soman, and P. Poornachandran, “Evaluating Shallow and Deep Neural Networks for Network Intrusion Detection Systems in Cyber Security,” *2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018*, Oct. 2018, doi: 10.1109/ICCCNT.2018.8494096.
- [5] “ENISA Threat Landscape Report 2018 — ENISA.” <https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018> (accessed Jun. 14, 2022).
- [6] “ENISA Threat Landscape Report 2017 — ENISA.” <https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2017> (accessed Jun. 14, 2022).
- [7] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, “Network Intrusion Detection for IoT Security Based on Learning Techniques,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 3, pp. 2671–2701, Jul. 2019, doi: 10.1109/COMST.2019.2896380.
- [8] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, “Intrusion detection model using machine learning algorithm on Big Data environment,” *Journal of Big Data*, vol. 5, no. 1, pp. 1–12, Dec. 2018, doi: 10.1186/S40537-018-0145-4/FIGURES/5.
- [9] A. Hussain and P. Kumar Sharma, “Efficient Working of Signature Based Intrusion Detection Technique in Computer Networks,” *International Journal of Scientific Research in Computer Science*,

- [10] R. Magán-Carrión, D. Urda, I. Díaz-Cano, and B. Dorronsoro, “Towards a Reliable Comparison and Evaluation of Network Intrusion Detection Systems Based on Machine Learning Approaches,” *Applied Sciences* 2020, Vol. 10, Page 1775, vol. 10, no. 5, p. 1775, Mar. 2020, doi: 10.3390/APP10051775.
- [11] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, Oct. 2020, doi: 10.1002/ETT.4150.
- [12] N. Thapa, Z. Liu, D. B. Kc, B. Gokaraju, and K. Roy, “Comparison of Machine Learning and Deep Learning Models for Network Intrusion Detection Systems,” *Future Internet* 2020, Vol. 12, Page 167, vol. 12, no. 10, p. 167, Sep. 2020, doi: 10.3390/FI12100167.
- [13] A. O. Alzahrani and M. J. F. Alenazi, “Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks,” *Future Internet* 2021, Vol. 13, Page 111, vol. 13, no. 5, p. 111, Apr. 2021, doi: 10.3390/FI13050111.
- [14] M. Injat, A. Moubayed, A. B. Nassif, and A. Shami, “Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1803–1816, Aug. 2020, doi: 10.1109/TNSM.2020.3014929.
- [15] “LUFlow Network Intrusion Detection Data Set | Kaggle.” <https://www.kaggle.com/datasets/mryanm/luflow-network-intrusion-detection-data-set> (accessed Jun. 17, 2022).
- [16] “GitHub - luciferreeves/Network-Intrusion-Detection: A Comprehensive Approach To Analysis and Detection of Emerging Threats due to Network Intrusion.” <https://github.com/luciferreeves/Network-Intrusion-Detection> (accessed Jun. 17, 2022).
- [17] Z. Zeng, W. Peng, and B. Zhao, “Improving the Accuracy of Network Intrusion Detection with Causal Machine Learning,” *Security and Communication Networks*, vol. 2021, 2021, doi: 10.1155/2021/8986243.
- [18] “A Gentle Introduction to the Rectified Linear Unit (ReLU).” <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/> (accessed Jun. 17, 2022).
- [19] “Rectified Linear Units (ReLU) in Deep Learning | Kaggle.” <https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook> (accessed Jun. 17, 2022).
- [20] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
- [21] “A Gentle Introduction to Dropout for Regularizing Deep Neural Networks.” <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/> (accessed Jun. 17, 2022).
- [22] H. Pratiwi *et al.*, “Sigmoid Activation Function in Selecting the Best Model of Artificial Neural Networks,” *Journal of Physics: Conference Series*, vol. 1471, no. 1, p. 012010, Feb. 2020, doi: 10.1088/1742-6596/1471/1/012010.
- [23] “Shiny.” <https://shiny.rstudio.com/> (accessed Jun. 17, 2022).
- [24] “Deep Learning (Neural Networks) — H2O 3.36.1.2 documentation.” <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html> (accessed Jun. 17, 2022).