Computer Communications 199 (2023) 113–125

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A deep learning technique for intrusion detection system using a Recurrent Neural Networks based framework

Sydney Mambwe Kasongo

Department of Industrial Engineering & The School for Data Science and Computational Thinking, Stellenbosch University, South Africa

ARTICLE INFO

Keywords:
Machine learning
Feature selection
Intrusion detection
Feature extraction

ABSTRACT

In recent years, the spike in the amount of information transmitted through communication infrastructures has increased due to the advances in technologies such as cloud computing, vehicular networks systems, the Internet of Things (IoT), etc. As a result, attackers have multiplied their efforts for the purpose of rendering network systems vulnerable. Therefore, it is of utmost importance to improve the security of those network systems. In this study, an IDS framework using Machine Learning (ML) techniques is implemented. This framework uses different types of Recurrent Neural Networks (RNNs), namely, Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU) and Simple RNN. To assess the performance of the proposed IDS framework, the NSL-KDD and the UNSW-NB15 benchmark datasets are considered. Moreover, existing IDSs suffer from low test accuracy scores in detecting new attacks as the feature dimension grows. In this study, an XGBoost-based feature selection algorithm was implemented to reduce the feature space of each dataset. Following that process, 17 and 22 relevant attributes were picked from the UNSW-NB15 and NSL-KDD, respectively. The accuracy obtained through the test subsets was used as the main performance metric in conjunction with the F1-Score, the validation accuracy, and the training time (in seconds). The results showed that for the binary classification tasks using the NSL-KDD, the XGBoost-LSTM achieved the best performance with a test accuracy (TAC) of 88.13%, a validation accuracy (VAC) of 99.49% and a training time of 225.46 s. For the UNSW-NB15, the XGBoost-Simple-RNN was the most efficient model with a TAC of 87.07%. For the multiclass classification scheme, the XGBoost-LSTM achieved a TAC of 86.93% over the NSL-KDD and the XGBoost-GRU obtained a TAC of 78.40% over the UNSW-NB15 dataset. These results demonstrated that our proposed IDS framework performed optimally in comparison to existing methods.

1. Introduction

Technologies such as Cloud computing, the Internet, Modern Industrial Control Systems, vehicular networks, etc. have rapidly evolved in recent years. These systems often work in symbiosis and handle large volumes of information through various communication infrastructures and complex communication networks such as 5G networks. As a result, many hackers and ill intentioned entities strive to find new ways to break into those computer systems by compromising the communication channels. Network intrusions are amongst some of the most dangerous security threats facing a number of organization today [1]. Therefore, Intrusion Detection Systems (IDSs) are implemented in order to secure computer systems and networks. An IDS is an artefact (hardware or software) that monitors a computer network's traffic for threats [2]. IDSs are categorized based on the detection technique they employ. The main classes of IDSs include: signature-based IDS (S-IDS), anomaly-based IDS (A-IDS) and Hybrid-inspired IDS (H-IDS). S-IDS is employed to detect attacks with signatures (patterns) that are existing in the IDS database. In contrast, A-IDS scans the network to identify and to learn which patterns (behaviours) constitute threats or intrusions. H-IDS is a combination of A-IDS and S-IDS [3].

In this research we present a Machine Learning (ML) based IDS that uses the Deep Learning (DL) paradigm. ML is a technique that gives computer programs the ability to learn tasks without being explicitly instructed to do so. In other words, ML based systems are capable of learning from experience [4]. ML is subdivided in the following types: Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL) [5]. RL is a ML technique whereby a computer program learns to execute certain tasks within a given environment which achieve a maximum reward. A UL based algorithm such as Clustering is concerned with data that is not labelled. On the other hand, a SL based ML algorithm predicts a label when provided with a list of inputs (dependent variables). In this instance, the data used for predictions is labelled. SL is mainly subdivided into Classification and Regression tasks. The solution to a classification problem is the prediction a discrete value. In contrast, the outcome of a Regression task is the prediction of a continuous value [6]. DL is a branch of

E-mail addresses: sydneybleuops@gmail.com, sydneyk@sun.ac.za.

https://doi.org/10.1016/j.comcom.2022.12.010
Received 14 January 2021; Received in revised form 24 October 2022; Accepted 5 December 2022
Available online 7 December 2022

ML that incorporates various methods whose structures are inspired from the inner workings of the neurons in the human brain [7]. There exists a number of DL based algorithms including Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs) etc. In this paper we propose a SL IDS based on the application of DL using different types of RNNs including: the Gated Recurrent Unit (GRU), the Long-Short Term Memory (LSTM) and Simple RNN algorithms. GRUs and LSTMs are modified types of Simple RNNs [8]. The advantage of using RNN based classifiers lies in the fact these types of DL networks are capable of temporarily memorizing previous states and apply them on the current computation. The RNN-IDS framework is implemented in conjunction with a feature selection method that is inspired from the Extreme Gradient Boosting (XGBoost) algorithm [9]. Moreover, to assess the efficiency of our IDS framework, we considered two IDS benchmark datasets, namely, the UNSW-NB15 and the NSL-KDD datasets [10,11]. The XGBoost algorithm is implemented through the datasets to extract the optimal feature vectors. This method differs from existing techniques because it uses an optimized Ensemble Tree technique that is capable to quickly generate a feature importance score for each feature present in a given dataset. The higher the feature importance score, the more critical a feature is. Moreover, Feature Selection has presented a lot of successes in increasing the performance of many SL based algorithms by reducing the dimension (size) of the feature space. The smaller and more optimal the feature space, the less complex a model becomes in terms of the training and the testing processes. The experiments conducted in this research have demonstrated that the XGBoost-RNN based IDSs perform optimally in comparison to other existing methods that are surveyed in this work. The main contributions of this paper are as follows:

- An XGBoost-based feature selection method is implemented on two IDS benchmark datasets.
- A training time analysis between LSTMs, GRUs and Simple RNNs is presented.
- RNNs variations are described in details.
- The XGBoost algorithm is known to be efficient and it allows for the optimization of arbitrary differentiable loss functions. Therefore, we use this algorithm for an optimized feature importances generation. Moreover, the XGBoost-RNN-IDS is implemented in three flavours of RNNs, namely, GRU, LSTM and Simple RNN. Further, a thorough performance analysis of the XGBoost-RNN-IDS framework is conducted using different performance metrics. The XGBoost-RNN-IDS is also compared to other existing IDS frameworks.

The reminder of this paper is structured as follows: Section 2. provides a survey of related studies. Section 3 gives the details about Simple RNNs, LSTMs and GRUs. Section 4 explains in detail all the datasets that are used in this paper. Section 5. introduces the structure of the proposed the RNN-IDS framework. Section 6. highlights the importance of the Feature Normalization procedure. Section 7 presents the XGBoost algorithm. Moreover, the XGBoost feature selection method is implemented and 17 features of the UNSW-NB15 were picked and 22 attributes of the NSL-KDD were selected. Section 8. is all about the performance evaluation of the proposed framework. Details about the hardware and software system used in our experiments are provided. The considered performance metrics are highlighted and the experimental results are presented and discussed. Section 9. concludes this research.

2. Related work

In this section, an account of previous studies about ML based IDSs that were implemented using various Deep Learning methodologies is presented.

In [12], the authors presented a ML based IDS using LSTM coupled with the multivariate correlation analysis (MCA). The MCA-LSTM used the Information Gain (IG) method as a feature selection approach. Firstly, the model picks a subset of features. Furthermore, the selected subset is transformed into a triangle area map matrix (TAM). Finally, The TAM is used by the LSTM algorithm to predict intrusions. In order to measure the performance of their model, the authors used the NSL-KDD and UNSW-NB15 datasets. The experimental results demonstrated that the MCA-LSTM achieved a test accuracy of 82.15% for the 5-way classification using the NSL-KDD. In the instance of the UNSW-NB15, the MCA-LSTM obtained a test accuracy of 77.74% for the 10-way classification task. For the binary classification problem, the MCA-LSTM achieved an accuracy 80.52% using the NSL-KDD dataset and 88.11% using the UNSW-NB15 dataset. Although these results were superior to other methods, the authors did not study the impact of the datasets size and also, they did not consider a wide array of performance metrics such as the F1-Score. Moreover, they did not highlight the strategies used to insure that their models were not over-fitting during the training process.

The researchers in [13] implemented a Deep Learning based IDS using RNNs. In this research, the authors used simple RNNs. Their framework was structured as follows: a training set was fed into a data processing block that was tasked with the transformation of categorical data into numeric inputs. Moreover, all the input were normalized using a scaling function. Furthermore, the data processing block would send information to the training block for training and model building. The dataset used in this study is the NSL-KDD dataset. For the performance evaluation, the authors considered the accuracy obtained through the test data as the main criterion for an optimal model. The results showed that for the binary classification scheme, the RNN-IDS achieved a test accuracy of 83.28% (The training time for this model was 5516 s). In contrast, the RNN-IDS obtained 81.29% (the training time was 11444 s) for the 5-way classification task. This research did not implement any feature reduction technique that could possibly increase the performance of the RNN-IDS as well as reduce the training and testing times.

In [14] a DL approach for wireless intrusion detection (WIDS) using a Feed-Forward Deep Neural Network (FFDNN) was implemented. The FFDNN-WIDS was equipped with a wrapper-based feature extraction module using the Extra Trees method. The aim was to produce an optimal input subset that would be fed to the FFDNN classifier so as to detect network intrusions. The authors considered the UNSW-NB15 and the AWID datasets for evaluation purposes. Unlike the UNSW-NB15 which is a general purpose dataset, the AWID is a dataset that is tailored for wireless networks traffic [15]. In the case of UNSW-NB15, the feature selection algorithm extracted 22 inputs whereas in the instance of the AWID, 26 features were selected. The authors conducted multiple experiments involving the binary and the multiclass classification problems. The results showed that the FFDNN (UNSW-NB15) achieved test accuracies of 87.10% for the 2-way classification task and 77.16% for the 10-way classification problem. Moreover, using the AWID dataset, the FFDNN obtained a test accuracy of 99.66% for the 2-way classification problem and a test score of 99.77% for the multi-way classification task. Although the research showed viable results, it needs to be benchmarked against other ML algorithms besides those mentioned by the authors.

A.H. Miraza and S. Cosan [16] developed an IDS framework using Autoencoders for input selection and the LSTM method for various attacks predications based on a variable length input space generated by the Autoencorders. For the performance evaluation of their system, the researchers used the ISCX IDS 2012 dataset. The F1-score and AUC score performance metrics were selected for this study. For contrasting purposes, the authors also considered the Bi-LSTM and GRU algorithms. After several experiments, the results showed that the Autoencoder-LSTM method outperformed other algorithms. The Autoencoder-LSTM achieved a F1-score of 85.38% and a AUC score of 0.9512. This research lacked clarity in terms of the performance metrics such as the accuracy on training and testing data. Moreover, there was no clear description of the dataset used in this work.

In [17], two deep learning techniques for intrusion detection were proposed. Firstly, the authors introduced an IDS based on the LSTM method. Secondly, the researchers presented the Convolutional Neural Network (CNN) [18] together with the LSTM algorithm. The NSL-KDD dataset was utilized as a basis for their experiments. In their data processing step, the authors used the input scaling method to cater for the variation in attributes values. The authors used the accuracy on the test data (KDD Test+) as the main metric to assess the performance of their models. Furthermore, the obtained results were also compared with those achieved by the Simple RNNs. The proposed LSTM got accuracies of 74.770% and 68.780% for the binary and multiclass classification task, respectively. The CNN-LSTM obtained a test accuracy of 79.37% for the binary classification scheme and 70.13% for the 5-way classification. In contrast, the RNN obtained 68.55% for the 2-way classification and 64.67% for the multiclass configuration. The authors of this research concluded that the implementation of an adequate feature extraction method could improve the results obtained by their proposed methods.

Z. Li and Z.A. Qin [19] developed an IDS that uses a semantic based algorithm in order to parse the input data. The LSTM algorithm is used to conduct the predictions. The author employed the NSL-KDD dataset in the experimental processes. Furthermore, the accuracy obtained on the test data was used as the main performance metric. A secondary performance metric used in this research was the F1-score. The Semantic-LSTM was compared to methods such as Naive Bayes, Random Forest, Multilayer Perceptron and Support Vector Machine. After a number of experiments, the Semantic-LSTM achieved an accuracy of 82.21% on the test data (NSL-KDD Test+) and a F1-score of 82.27%. The authors concluded that this research was not enough and that more variants of RNNs should be explored.

In [20], an IDS using the LSTM and GRU methods was presented. These methods were implemented on the NSL-KDD dataset, the BC-NET dataset as well as the Border-Gate-Way (BGW) dataset. With the purpose of assessing the performance of the studied algorithms, the researchers considered the accuracy obtained on the test data as well as the F1-score metric. With an emphasis on the NSL-KDD, the experiments demonstrated that the LSTM method achieved a test score of 82.78% and a F1-score equating to 83.34%. With regards to the GRU algorithm, the following scores were obtained: 82.87% and 83.05% for the test accuracy and the F1-score, respectively.

In [21] an IDS based on Deep Learning Hybrid Techniques was proposed. The authors investigated various ML approaches and they evaluated them using the NSL-KDD dataset. The LSTM method was one of the algorithms which was implemented. In this research, the authors did not consider a feature extraction method. Furthermore, to assess the efficiency of the algorithms presented in this work, the researchers considered the False Negative Rate (FNR), the accuracy obtained on the test subset and the False Positive Rate (FPR) as the performance metrics. In the instance of the LSTM-IDS, the results demonstrated that the most efficient LSTM configuration achieved a FPR of 3.00 %, a FNR of 26.00% and a test accuracy score of 82.00%.

Tang et al. [22] implemented an IDS using Deep Learning for Software Defined Networks (SDN). SDN is a paradigm that is destined to revolutionize the Internet. In this research, the authors used Deep Neural Network (DNN) as the ML method in their IDS system. Moreover, the researchers used the NSL-KDD dataset in order to evaluate the performance of the DNN-IDS. Moreover, the author manually selected a subset of 6 features that were contemplated to be of valuable importance for SDNs. The accuracy achieved by a DNN on the test data was the main performance metric that was considered. The experimental results demonstrated that the DNN-IDS achieved an accuracy of 75.75%.

In [23], an IDS using a Genetic Algorithm (GA) coupled with the Logistic Regression (LR) was proposed. The GA-LR was evaluated using the KDDCup99 and the UNSW-NB15 datasets. The GA-LR algorithm was used as the wrapper-based feature selection method and the Decision Tree (DT) classifier was utilized as the ML method. The attributes selection process resulted in an optimal feature vector containing 20 features for the UNSW-NB15 and 18 features for the KDDCup99. The authors considered the accuracy on the test data and the False Alarm Rate (FAR) as the performance metrics. In the instance of the KDDCup99 dataset, the GA-LR achieved an accuracy of 99.90% and a FAR of 0.105%. For the UNSW-NB15, the GA-LR obtained an accuracy of 81.42% and a FAR of 6.39%.

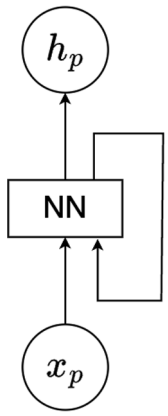

Fig. 1. A standard RNN.

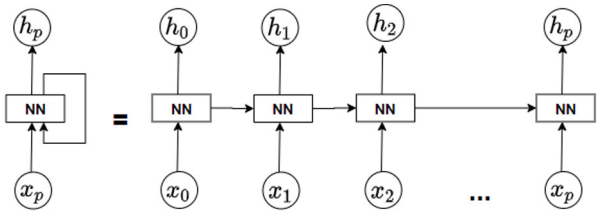

Fig. 2. A unrolled standard RNN.

Mebawondu et al. [24] proposed an IDS using a feature extraction method based on the Gain Ratio (GR) measure and a classification approach using Artificial Neural Networks (Multi-Layer Perceptrons). In this study, the authors used the UNSW-NB15 dataset to assess the efficiency of their proposed IDS framework. The feature extraction procedure lead to the selection of 30 attributes of the UNSW-NB15. The experimental processes were then carried out using the reduced feature vector and the results demonstrated that the GR-ANN-MLP achieved a test accuracy of 76.96% and a Precision of 79.80%.

3. Recurrent Neural Networks

In comparison to basic neural networks such as Multilayer Perceptrons (MLPs) [25], Recurrent neural networks (RNNs) are not restricted to processing information in one direction only. RNNs are able to loop through different layers and they are also capable of temporarily memorizing information for later use [26]. The structure of a standard RNN (sRNN) or Simple RNN is depicted in Fig. 1. NN denotes a standard neural network, x_p represents the input and h_p is the output.

By definition, RNNs are considered deep neural networks because there are various layers through which information is processed. As described in Fig. 2, a standard sRNN is unrolled to illustrate the depth of the RNN construct. Although sRNNs are effective in performing various prediction tasks, they do, however, have a problem of a vanishing gradient. To solve this issue, other types of RNNs such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) were engineered [27].

The training process of the sRNN model is conducted using a forward–backward propagation (FFP) technique. This method is often

employed in MLPs. In the instance of sRNN, the FFP happens through time steps [28]. The FFP process is formulated as follows:

Let x_p denote the input to sRNN model. The weights of the connection between the input x_p and the first (current) hidden layer is represented by w_{hx}. The weight of the link connecting the current hidden layer with the following hidden layer is denoted by w_{hh}. The weight between the last hidden layer with its matching output layer is represented by w_{hy}. Let h_p denote the output at time p. The biases added to the connections between the hidden layer and the output layer are represented by b_h and b_y, respectively. A sRNN is mathematically constructed as follows:

$$h_p = g(h_{p-1}, x_p) \tag{1}$$

$$h_p = g\left([w_{hx} \cdot x_p + w_{hh} \cdot x_{p-1}] + b_h\right) \tag{2}$$

$$\bar{y}_p = g\left([w_{hx} \cdot x_p + w_{hh} \cdot x_{p-1}] + b_h\right) \tag{3}$$

where $\bar{y}_p$ is the predicted result.

The sRNN is trained in sequence. At each step p, the error (E) between the prediction and the actual (true) output values is computed. In ML terminology, E is called *Loss*, L. The ultimate aim of the training procedure is to reduce L. In other words, a good model will possess a minimal L. For a given dataset, the L generated by the training of one batch of data at time, p, is calculated in the following expression:

$$L(\bar{y}_p, y) = \sum_{p=1}^{N} L(\bar{y}_p - y_p) \tag{4}$$

where N denotes the maximum training time. sRNN have a reputation of suffering from a vanishing gradient issue or an explosion of the gradient during the training procedure. These problems always occur as the size of a given dataset expands and as N increases. The following paragraphs provides more details about the issue pertaining to the gradient.

Let L represent the E generated by a batch, B, during a maximum training time N. Eq. (5) denote the computation of the partial derivative of the loss with regards to the associated weight, W.

$$\frac{\partial L}{\partial W} = \sum_{p=1}^{N} \frac{\partial L_p}{\partial W} \tag{5}$$

By applying the chain rule to the expression in (5), $\frac{\partial L}{\partial W}$ is reformulated as follows:

$$\frac{\partial L}{\partial W} = \sum_{p=1}^{N} \frac{\partial L_p}{\partial y_p} \frac{\partial y_p}{\partial h_p} \frac{\partial h_p}{\partial y_n} \frac{\partial y_n}{\partial W} \tag{6}$$

The Eq. (6) represents the derivative of a given hidden state which stores information at time p. This derivative is connected to the state of time n.

The Eq. (7) is the derivative of a hidden state that keeps information at time p that is directly linked to the hidden state of the previous time n. Moreover, the expression in (7) contains the Jacobians matrix [26] in connection to time p and n, $\frac{\partial h_p}{\partial h_n}$.

$$\frac{\partial h_p}{\partial h_n} = \frac{\partial h_p}{\partial h_{p-1}} \frac{\partial h_{p-1}}{\partial h_{p-2}} \cdots \frac{\partial h_{n+1}}{\partial h_n} = \prod_{i=n+1}^{p} \frac{\partial h_i}{\partial h_{i-1}} \tag{7}$$

The Eqs. (5) and (7) generate an Eigen Decomposition Vector (EDV) that is expressed as follows:

$$EDV = W^T diag\left[g'(h_{p-1})\right] \tag{8}$$

The EDV generates these eigenvalues: $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_n$ contingent on the following constraint: $|\lambda_1| > |\lambda_2||\lambda_3| \ldots |\lambda_n|$. All the eigenvalues are correlated to their corresponding eigenvectors: $ev_1, ev_2, ev_3 \ldots, ev_n$. The instance of a *vanishing* gradient occurs when the largest $\lambda_i < 1$. The gradient *explodes* in the case whereby $\lambda_i > 1$. The *vanishing* and *exploding* gradient problems are addressed by the introduction of LSTMs [29]

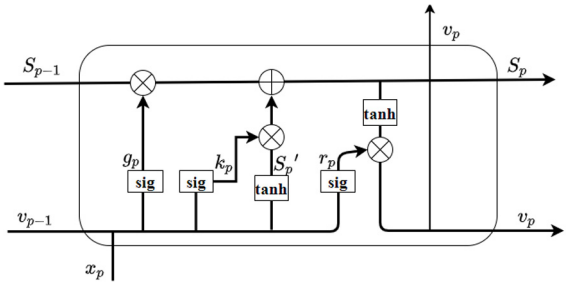

Fig. 3. LSTM unit.

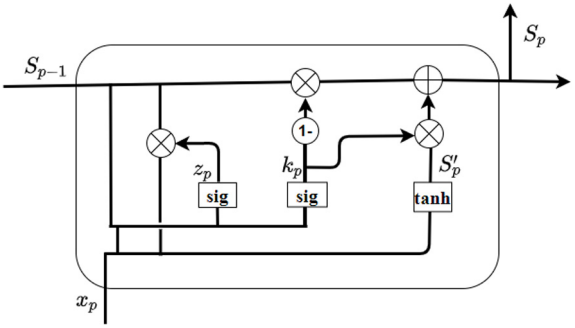

Fig. 4. GRU unit.

and GRUs [30]. The structure of the LSTM used in this study is shown in Fig. 3 and that of the GRU Unit is depicted in Fig. 4. The expressions defined in (9) and (10) provide a mathematical formulation of the flow of information within a LSTM and GRU, respectively.

$$\textbf{LSTM Formulation:} \begin{cases} g_p = \sigma(W_g \,.\, [v_{p-1}, x_p] + b_g) \\ k_p = \sigma(W_k \,.\, [v_{p-1}, x_p] + b_p) \\ S_p{}' = tanh(W_s \,.\, [v_{p-1}, x_p] + b_s) \\ S_p = g_p * S_{p-1} + p_t * S_p{}' \\ r_p = \sigma(W_r \,.\, [v_{p-1}, x_p] + b_r) \\ v_p = r_p * tanh(S_p), \end{cases} \tag{9}$$

where S denotes the cell state. The following activation functions are used in the LSTM unit: hyperbolic tangent, $tanh(a) = \frac{1 - e^{-2a}}{1 + e^{-2a}}$ and the Sigmoid, $\sigma(a) = \frac{a}{1 + e^{-a}}$. x represents the input vector. v denotes the output vector. W and b represent the weights and their related biases.

$$\textbf{GRU Formulation:} \begin{cases} k_p = \sigma(W_k.[S_{p-1}, x_p] + x_p) \\ z_p = \sigma(W_z.[S_{p-1}, x_p]) \\ S_p{}' = tanh(W.[z_p * S_{p-1}, x_p]) \\ S_p = (1 - k_p) * S_{p-1} + k_p * S_p{}', \end{cases} \tag{10}$$

where x is the input vector. S_p is the computed prediction. k_p denotes the update function. The corresponding weight is defined by W. As in the LSTM in Eq. (9), the GRU uses the hyperbolic tangent as well as the Sigmoid activation functions in order to squash the information.

4. Datasets

4.1. NSL-KDD

The first dataset considered in this study is the NSL-KDD dataset [31]. This dataset is one of the most popular ones used in the evaluation of IDS frameworks. Besides *normal* network traffic, the NSL-KDD contains these four types of intrusions: *DoS*, *U2R*, *R2L* and *Probe*. In this study, two subsets of the NSL-KDD dataset are considered, namely, the

Table 1
NSL-KDD attributes description.

No.	Attribute	Format	No.	Name	Format
f1	duration	numeric	f22	is_guest_login	numeric
f2	protocol_type	categorical	f23	count	numeric
f3	service	categorical	f24	srv_count	numeric
f4	flag	categorical	f25	serror_rate	numeric
f5	src_bytes	numeric	f26	srv_serror_rate	numeric
f6	dst_bytes	numeric	f27	rerror_rate	numeric
f7	land	numeric	f28	srv_rerror_rate	numeric
f8	wrong_fragment	numeric	f29	same_srv_rate	numeric
f9	urgent	numeric	f30	diff_srv_rate	numeric
f10	hot	numeric	f31	srv_diff_host_rate	numeric
f11	num_failed_logins	numeric	f32	dst_host_count	numeric
f12	logged_in	numeric	f33	dst_host_srv_count	numeric
f13	num_compromised	numeric	f34	dst_host_same_srv_rate	numeric
f14	root_shell	numeric	f35	dst_host_diff_srv_rate	numeric
f15	su_attempted	numeric	f36	dst_host_same_src_port_rate	numeric
f16	num_root	numeric	f37	dst_host_srv_diff_host_rate	numeric
f17	num_file_creations	numeric	f38	dst_host_serror_rate	numeric
f18	num_shells	numeric	f39	dst_host_srv_serror_rate	numeric
f19	num_access_files	numeric	f40	dst_host_rerror_rate	numeric
f20	num_outbound_cmds	numeric	f41	dst_host_srv_rerror_rate	numeric
f21	is_host_login	numeric			

Table 2
NSL-KDD data subsets: Training, validation and test.

Dataset	Normal	DoS	Probe	R2L	U2R	Total
KDDTrain	67343	45927	11656	995	52	125973
KDDTrain+	50494	34478	8717	749	42	94480
KDDVal)	16849	11449	2939	246	10	31493
KDDTest+ Full	9711	7458	2754	2421	200	22544

NSL-KDD-Train and the NSL-KDDTest+. The NSL-KDD-Train is further divided into the following two partitions: the NSL-KDD-Train+ and the NSL-KDDVal. The first subset constitutes 75% of the NSL-KDD-Train and it is used during the training process. The later represents 25% of the NSL-KDD-Train and it is utilized for the validation process. The validation process will ensure that the models used in this study are not subject to overfitting during the training step [32]. The same procedure is considered for the UNSW-NB15 dataset that is described in the next subsection. The NSL-KDDTest+ is employed to test the validated models. Table 1 lists all the features present in the NSL-KDD dataset whereby 3 attributes are categorical and the rest are of numerical nature. Table 2 gives a detailed account of the values present in each data subsets.

4.2. UNSW-NB15

The UNSW-NB15 [33] is the second dataset considered in this research in order to assess the proposed framework. The UNSW-NB15 contains 42 attributes that include 3 categorical inputs and 39 numerical inputs. The numerical inputs have the following data format (types): *binary*, *integer* and *float*. Moreover, the UNSW-NB15 incorporates two data subsets, one for the training procedure and the other for the testing process. Similarly to the NSL-KDD dataset, we have separated the UNSW-NB15 training set into the following two partitions: the UNSW-NB15Train+ (75% of the original training) that is used for training the models and the UNSW-NB15Val (25% of the original training) used to validate the trained models. Additionally, the UNSW-NB15-Test is utilized for testing the validated models. This separation of subsets ensures that none of the models trains on the validation or the test data subsets. This ensures that the obtained results are not biased or compromised by any data leakage. The UNSW-NB15 comprises these nine classes of attacks: Exploits, Shellcode, Fuzzers, Generic, Worms, DoS, Reconnaissance, Analysis, Backdoor and Reconnaissance. Table 3 lists the attributes of the UNSW-NB15 and Table 4 provides a detailed description of the attacks distribution of the UNSW-NB15 subsets.

Table 3
UNSW-NB15 attributes description.

No.	Attribute	Format	No.	Attribute	Format
f1	dur	float	f22	dtcpb	integer
f2	proto	categorical	f23	dwin	integer
f3	service	categorical	f24	tcprtt	float
f4	state	categorical	f25	synack	float
f5	spkts	integer	f26	ackdat	float
f6	dpkts	integer	f27	smean	integer
f7	sbytes	integer	f28	dmean	integer
f8	dbytes	integer	f29	trans_depth	integer
f9	rate	float	f30	response_body_len	integer
f10	sttl	integer	f31	ct_srv_src	integer
f11	dttl	integer	f32	ct_state_ttl	integer
f12	sload	float	f33	ct_dst_ltm	integer
f13	dload	float	f34	ct_src_dport_ltm	integer
f14	sloss	integer	f35	ct_dst_sport_ltm	integer
f15	dloss	integer	f36	ct_dst_src_ltm	integer
f16	sinpkt	float	f37	is_ftp_login	binary
f17	dinpkt	float	f38	ct_ftp_cmd	integer
f18	sjit	float	f39	ct_flw_http_mthd	integer
f19	djit	float	f40	ct_src_ltm	integer
f20	swin	integer	f41	ct_srv_dst	integer
f21	stcpb	integer	f42	is_sm_ips_ports	binary

Table 4
UNSW-NB15 data subsets: Training, validation and test.

Attack type	UNSW-NB15	UNSW-NB15-75	UNSW-NB15-25	UNSW-NB15-TEST
Normal	56000	41911	14089	37000
Generic	40000	30081	9919	18871
Exploits	33393	25034	8359	11132
Fuzzers	18184	13608	4576	6062
DoS	12264	9237	3027	4089
Reconnaissance	10491	7875	2616	3496
Analysis	2000	1477	523	677
Backdoor	1746	1330	416	583
Shellcode	1133	854	279	378
Worms	130	99	31	44

5. The proposed IDS framework

Fig. 5 depicts the IDS framework that is proposed in this study. The initial phase consists in collecting the necessary data for the models building. In the case of this research, two datasets were considered, namely, the NSL-KDD and the UNSW-NB15. The second layer of the proposed architecture is the Data Processing and Feature Extraction (Selection) phase. In this step, a particular dataset is normalized to ensure that all the categorical attributes are correctly encoded and that all the numerical features are normalized (Section 6). The XGBoost algorithm (Section 7) is then applied once the dataset has been cleaned and normalized. This process generates a vector containing the Feature Importance (FI) values and an optimal feature subset is picked based on an empirically selected FI threshold. The third layer of the architecture is the Model Building phase. In this phase, there are 3 major operations that occur independently, namely, Training, Validation and Testing. Each of this steps is evaluated using the performance criteria that are set out in Section 8. The final step is the selection of the optimal model required to conduct intrusion detection.

Moreover, Fig. 6 depicts the configuration of the RNN, LSTM and GRU networks that are investigated in this research. The first layer is the input layer which feeds into a construct of RNN/LSTM/GRU deep layers. The information from the deep layers is then computed through a Dense NN layer (This can be a single layer Artificial Neural Network (ANN) or a Multilayer Perceptron (MLP). Finally, the predictions are computed through a $Softmax$ (Eq. (11)) for the multiclass classification configurations, or the Rectified Linear Unit ($ReLU$) (Eq. (12)) and the Sigmoid function for the binary classification setup [34]. The $Softmax$

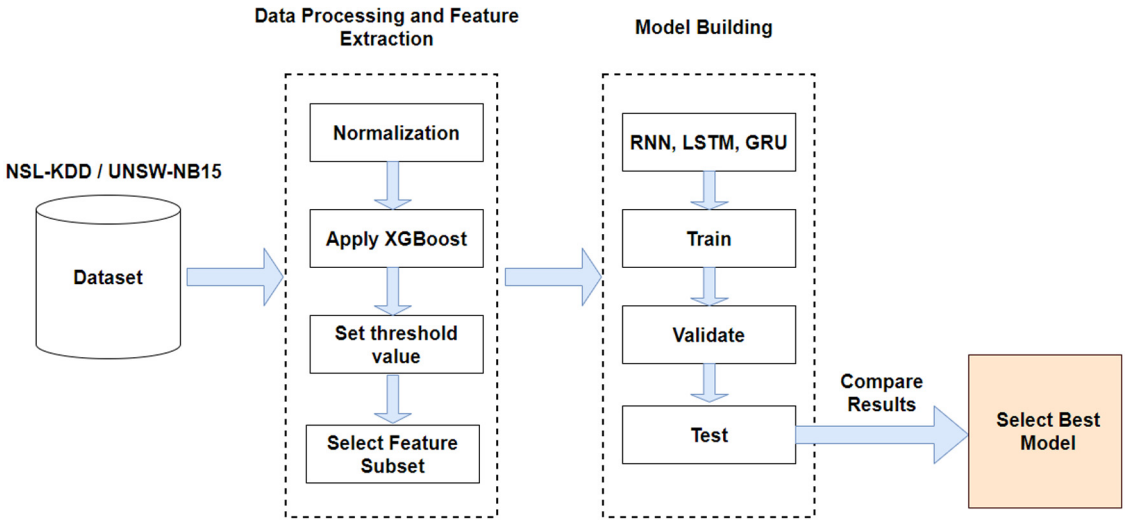

Fig. 5. The proposed RNN based IDS framework.

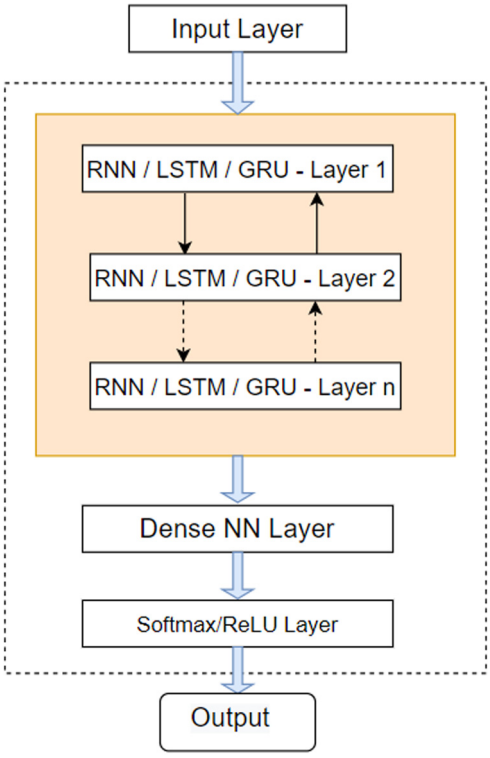

Fig. 6. RNN/LSTM/GRU structure.

activation function returns a vector that contains values (probabilities) that add up to 1 [35]. The highest value represents the prediction. The $ReLU$ on the other hand, returns 0 for any negative input or otherwise it returns the input. The Sigmoid, $Sigmoid(z) = \frac{1}{1+e^{-z}}$ function return a value between −1 and 1.

$$Softmax(z)_k = \frac{e^{z_k}}{\sum_{i=1}^{N} e^{z_i}} \tag{11}$$

$$ReLU(z) = max(0, z) \tag{12}$$

6. Feature normalization

Feature normalization or Feature Standardization is a critical aspect in the Data Processing and Feature Extraction phase of the proposed architecture. This is due to the fact most of the datasets features contain values that are not within the same range. Some are very large and some are very small or are zeros. In our study, we mainly used NNs (RNN Types). The performance of neural networks can be negatively affected by features with large values. Therefore, the normalization process has proven to increase the performance of NNs by reducing the convergence rate during the training procedure [36]. There are a number of ways a dataset can be normalized such as: Z-Score scaling, log normalization, decimal scaling, Min–Max scaling, etc. [37]. Moreover, selecting which technique to use is highly dependent on the application. However, in this study, we picked the Min–Max scaling method described Eq. (11).

$$V_{norm} = \frac{V - V_{min}}{V_{max} - V_{min}} \tag{13}$$

Let $X(x_1, \ldots, x_i)$, $1 < i < I$, be a feature space of a data subset J where I is the highest number of its instances. The Min–Max normalization of J is conduced as elaborated in Algorithm 1.

Algorithm 1 Pseudo Code for Min–Max Scaling

Input: $X(x_1, ..., x_i)$, $1 < i < I$
Output: $X_{trans}(x_1^{trans}, ..., x_n^{trans})$:
  for k from 1 to I **do**
    if (x_i a nominal attribute) **then**
      Step 1: Import and Instantiate a $labelEncoder$ object
      Step 2: Fit and Transform the $labelEncoder$
      Step 3: Save the $labelEncoder$ encoded attributes in new Panda series.
      Step 4: Apply Min-Max Scaling: $x_i^{trans} = \frac{x_i - (x_i)_{min}}{(x_i)_{max} - (x_i)_{min}}$
    end if
    Step 1: $x_i^{trans} = \frac{x_i - (x_i)_{min}}{(x_i)_{max} - (x_i)_{min}}$
  end for

7. XGBoost feature selection

The Extreme Gradient Boosting (XGBoost) approach or sometimes referred to as the Tree Boosting Algorithm is a ML technique that allows for the Tree algorithm to be boosted. XGBoost is similar to techniques such as the Extra Tree (ET) or the Random Forest (RF) algorithms. However, it differs with RF and ET in the fact the Trees in the XGBoost are not independent from each other but rather, each new Tree complements the existing ones [9,38–40].

Let $G = \{(x_k, y_k) : k = 1...p, x_k \in \mathbb{R}^q, y_k \in \mathbb{R}\}$ denote a given dataset where there are p records composed of q features denoted by y. Let $\hat{y}_k$

be the outcome of the ensemble of Trees model expressed as follows:

$$\hat{y}_k = \vartheta(x_k) = \sum_{i=1}^{I} f_i(x_k) \tag{14}$$

where f_i denotes a regression tree and $f_i(x_k)$ is the score value attributed to the ith tree of the kth instance in the dataset. The goal is to minimize the expression in Eq. (13).

$$E(\vartheta) = \sum_{n} l(y_i, \hat{y}_k) + \sum_{m} \Phi(f_i) \tag{15}$$

where l is represents the loss associated with loss function E. Moreover, Φ is used to penalize Eq. (13) and therefore, it reduces the model's complexity. Φ is defined as follows:

$$\Phi(f_i) = \sigma Q + \frac{1}{2}\upsilon \|W\|^2 \tag{16}$$

where $sigma$ and $upsilon$ regularize the number of tree leaves Q and size of their respective weights W. The aim of using Φ is to ensure there is no model overfitting during the tree boosting procedure.

The minimization process occurs in iteration. Hence, at the tth iteration, f_t is added to the objective expression as follows:

$$E^t = \sum_{i=k}^{p} l((y_k, (\hat{y}_k)^{t-1} + f_t(x_k))) + \Phi(f_t) \tag{17}$$

In order to simplify the above expressions (the Loss), the Taylor expansion is applied as follow:

$$E_{split} = \frac{1}{2}\left[\frac{(\sum_{k\in\Gamma_L} g_k)^2}{\sum_{k\in\Gamma_L} h_k + \sigma} + \frac{(\sum_{k\in\Gamma_R} g_k)^2}{\sum_{k\in\Gamma_R} h_k + \sigma}\right] - \frac{1}{2}\left[\frac{(\sum_{k\in\Gamma} g_k)^2}{\sum_{k\in\Gamma} h_k + \sigma}\right] - \upsilon \tag{18}$$

where Γ denotes the group of instances in the current node. τ_R are the instances on the right side of the node and τ_L on the left wing of the node. Moreover, g_k and h_k are the gradients of the Loss expression and they are defined as follows: $g_k = \partial_{\hat{y}^{(t-1)}} l(y_k, \hat{y}^{(t-1)})$ and $h_k = \partial^2_{\hat{y}^{(t-1)}} l(y_k, \hat{v}^{(t-1)})$.

In the instance of our study, we use the XGBoost to generate the FI of each feature. In a bid to achieve that, we utilize the XGBClassifier present in the xgboost Python library [41]. Algorithm 2 outlines the steps to follow in order to generate the FI scores of each attributes in a given dataset. Moreover, the procedure also extracts and optimal input vector containing only the features with the highest FI with regards to the FI threshold (FI_{th}). This value is empirically selected and varies from dataset to dataset. In this study, an FI that shows a significant decrease between its value and that of the next FI is selected as the threshold value. The rule is set as follows: if $FI_{next} < \eta * FI_{current}$): $FI_{current} = FI_{th}$ where $\eta = 0.35$.

We applied Algorithm 1 and Algorithm 2 to the NSL-KDD dataset as well as the UNSW-NB15 in order to get their respective FI scores. Table 5 shows the FI scores of the NSL-KDD dataset whereby the feature f5 (src_bytes) is the most important attribute with a score of 0.391409. Moreover, FI_{th}= 0.001741 and based on this score, 22 optimal attributes are selected (Table 6).

Table 6 provides the details about the FI scores that were obtained after running Algorithm 1 and Algorithm 2 over the UNSW-NB15. In this is instance, the feature f10 (sttl) is the most important attribute with a score of 0.803374. Additionally, the FI_{th}= 0.000526.

Algorithm 2 Input variables Selection using XGBoost

Input: $X_{norm}(x_1^{norm}, ..., x_n^{norm})$
Input: y, the target vector.
Output: X_{opt} : the optimal input variables vector
Step 1: Declare a list X_{trans}
Step 2: Declare a list L_{fi}.
Step 3: Instantiate and call $GradientBoostingClassifier$ object as $xgb\_classifier$ using X_{norm} and y
Step 4: Train and fit the $xgb\_classifier$
Step 5: Calculate the importance factors and load them into X_{trans}
Step 6: Set the FI threshold FI_{th}
Step 7:
for $item$ in X_{trans} **do**
  if $(FI(item) \geq FI_{th})$ **then**
    append $FI(item)$ into L_{fi}
  end if
end for
Sort in descending order: L_{fi}
Step 8: Use L_{fi} to generate X_{opt}

Table 5
NSL-KDD feature importance values.

No.	Attribute	Format	FI
f5	src_bytes	numeric	0.391409
f29	same_srv_rate	numeric	0.192999
f38	dst_host_serror_rate	numeric	0.075419
f35	dst_host_diff_srv_rate	numeric	0.050147
f37	dst_host_srv_diff_host_rate	numeric	0.049420
f23	count	numeric	0.038715
f2	protocol_type	categorical	0.036734
f30	diff_srv_rate	numeric	0.035528
f6	dst_bytes	numeric	0.028185
f33	dst_host_srv_count	numeric	0.024700
f25	serror_rate	numeric	0.011800
f8	wrong_fragment	numeric	0.010289
f13	num_compromised	numeric	0.010212
f3	service	categorical	0.009323
f36	dst_host_same_src_port_rate	numeric	0.007717
f10	hot	numeric	0.005777
f17	num_file_creations	numeric	0.005275
f40	dst_host_rerror_rate	numeric	0.004565
f32	dst_host_count	numeric	0.002844
f12	logged_in	numeric	0.002447
f34	dst_host_same_srv_rate	numeric	0.002380
f4	flag	categorical	0.001741
f14	root_shell	numeric	0.000568
f24	srv_count	numeric	0.000402
f1	duration	numeric	0.000384
f11	num_failed_logins	numeric	0.000252
f26	srv_serror_rate	numeric	0.000152
f16	num_root	numeric	0.000132
f18	num_shells	numeric	0.000125
f41	dst_host_srv_rerror_rate	numeric	8.62e−05
f15	su_attempted	numeric	7.5e−05
f27	rerror_rate	numeric	6.5-05
f22	is_guest_login	numeric	5.1e−05
f19	num_access_files	numeric	3.5e−05
f7	land	numeric	2.7e−05
f39	dst_host_srv_serror_rate	numeric	2.9e−06
f31	srv_diff_host_rate	numeric	8.5e−07
f28	srv_rerror_rate	numeric	8.0e−07
f9	urgent	numeric	0.0
f20	num_outbound_cmds	numeric	0.0
f21	is_host_login	numeric	0.0

8. Performance evaluation

8.1. Hardware and software systems

The Scikit-Learn [42] and the Keras [43] ML and DL Python Libraries were employed to carry out the experimental processes. In terms of hardware system, all the simulations were implemented on a DELL-153000 running the Windows 10 Operating System (OS) and a processor with the specifications as follows: Intel(R)-Core (TM) i7-8568U CPU-@1.80GHz-1.99 GHz.

8.2. Performance metrics

There exist a number of performance metrics that could be used to assess the efficiency of an ML based IDS framework, however, in the instance of this research, we used the following metrics: the accuracy(AC) (Eq. (19)) obtained on the test data (TAC), the AC obtained

Table 6
UNSW-NB15 feature importance values.

No.	Attribute	Format	FI
f10	sttl	numeric	0.803374
f41	ct_srv_dst	numeric	0.039387
f7	sbytes	numeric	0.037377
f27	smean	numeric	0.019878
f2	proto	categorical	0.018848
f32	ct_state_ttl	numeric	0.016783
f14	sloss	numeric	0.012008
f25	synack	numeric	0.010125
f36	ct_dst_src_ltm	numeric	0.007203
f28	dmean	numeric	0.007134
f31	ct_srv_src	numeric	0.006745
f3	service	categorical	0.006305
f35	ct_dst_sport_ltm	numeric	0.003717
f8	dbytes	numeric	0.002706
f15	dloss	numeric	0.001793
f4	state	categorical	0.001548
f24	tcprtt	numeric	0.001224
f34	ct_src_dport_ltm	numeric	0.000526
f9	rate	numeric	0.000503
f12	sload	numeric	0.000387
f39	ct_flw_http_mthd	numeric	0.000348
f30	response_body_len	numeric	0.000339
f29	trans_depth	numeric	0.000272
f36	ct_src_ltm	numeric	0.000250
f18	sjit	numeric	0.000221
f17	dinpkt	numeric	0.000182
f22	dtcpb	numeric	0.000147
f42	is_sm_ips_ports	numeric	0.000141
f11	dttl	numeric	0.000115
f1	dur	numeric	0.000103
f26	ackdat	numeric	9.8e−05
f6	dpkts	numeric	6.2e−05
f13	dload	numeric	5.2e−05
f21	stcpb	numeric	3.6e−05
f37	is_ftp_login	numeric	2.2e−05
f19	djit	numeric	1.658e−05
f33	ct_dst_ltm	numeric	1.1e−05
f16	sinpkt	numeric	2.8e−17
f5	spkts	numeric	0.0
f20	swin	numeric	0.0
f23	dwin	numeric	0.0
f38	ct_ftp_cmd	numeric	0.0

Table 7
Feature sets generated by the XGboost-based feature selection method.

Dataset	Count	Optimal features
NSL-KDD	22	f5, f29, f38, f35, f37, f23, f2, f30 f6, f33 ,f25,f8, f13, f3, f36 f10, f17, f40, f32, f12, f34, f4
UNSW-NB15	17	f10, f41, f7, f27, f2, f32, f14, f25 f36, f28, f31,f3, f35, f8, f15, f4, f24

on validation data (VAC), the F1-Score or F-Measure (F1S) (Eq. (20)) obtained on validation data and the training time in seconds (Ts) for each model that was used in the experiments.

$$AC = \frac{TN + TP}{FN + FP + TP + TN} \tag{19}$$

$$F1S = 2\frac{Pr.Rc}{Pr + Rc} \tag{20}$$

where Pr denotes the precision, $Pr = \frac{TP}{TP+FP}$ and Rc represents the recall, $Rc = \frac{TP}{TP+FN}$

The following are the parameters in Eq. (19), Pr and Rc:

- False Positive (FP) or Fall-out: legitimate/normal traces that is incorrectly labelled as intrusive.
- False Negative (FN): attack that is classified as normal.
- True Negative (TN): normal traffic that correctly labelled as legitimate.
- True Positive (TP): intrusion that is successfully labelled as attack.

Table 8
SimpleRNN binary classification — NSLKDD.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	S2	97.88%	81.36%	97.88%	43.40
15	3	R2	97.45%	79.20%	97.71%	56.94
30	3	S2	98.63%	80.79%	98.72%	80.66
30	3	R2	98.17%	83.33%	98.37%	59.62
40	3	R2	98.43%	81.01%	98.67%	93.61
40	3	S2	98.29%	82.09%	98.45%	54.32
60	3	R2	98.42%	82.87%	98.73%	78.11
60	3	S2	98.59%	83.70%	98.72%	78.19
90	3	R2	98.50%	80.65%	98.89%	90.63
90	3	S2	98.78%	82.70%	98.84%	107.32
120	3	R2	98.68%	81.44%	98.82%	105.48
120	3	S2	98.74%	83.15%	98.87%	104.88
150	3	R2	98.76%	82.82%	98.65%	102.25
150	3	S2	99.36%	82.91%	99.37%	111.41

8.3. Experiments and discussions

The experiments were carried out in two folds. In the first phase, the simulations were carried out using the NSL-KDD dataset using its reduced feature vector as listed in Table 7 (with 22 features). In Tables 8–19, NU is the Number of Hidden Units, HS represents the Hidden layers Size. Sf stands for $Softmax$, S is the $Sigmoid$ and R is the $ReLU$. The experiments conducted in the first phase included both the binary and the multiclass classifications tasks. We implemented 3 types of RNNs, namely: SimpleRNN, LSTM and GRU. The following were the hyperparameters:

- Binary classification: $loss$ = 'binary_crossentropy', $optimiser$ = 'sgd' (Stochastic Gradient Descent), initial $learning\_rate$ = 0.001 (adaptive), $input\_shape$ = (1, 22)
- Multi-classification: $loss$ = 'categorical_crossentropy', $optimiser$ = 'adam' (An extension of Stochastic Gradient Descent), initial $learning\_rate$ = 0.01 (adaptive), $input\_shape$ = (1, 22)

Additionally, we used the $Sigmoid$ and the $ReLU$ activation functions for the Dense layers. Moreover, for each model, we considered the training time in seconds. As depicted in Table 8 below, the results show that for the binary classification, the most optimal SimpleRNN model achieved a TAC of 83.70%, a F1S of 98.72%, a VAC of 98.59% and it was trained in 78.19 s.

The results for the LSTM method are listed in Table 9 whereby the most optimal model obtained a TAC of 88.13% using 150 LSTM units spread through 3 hidden layers and its training duration is 225.46 s. This model used the $ReLU$ activation function in its Dense layer. In the instance of the GRU algorithm (Table 10), the best classifier achieved a TAC of 85.70%, a F1S of 99.38% and a training time in 161.00 s. This model used the $Sigmoid$ activation function in its Dense layer. Moreover, it was configured using 90 GRU units in its hidden layers. Furthermore, in Fig. 7, we have conducted a comparison between the training times of the Simple RNN, LSTM and GRU algorithms. The results demonstrates that the Simple RNN trains faster than the GRU and LSTM algorithms which are more complex. However, the GRU approach trains faster then the LSTM method.

In Tables 11–13, the results for the multiclass classification scheme using the Simple RNN, LSTM and GRU are presented. For each of these algorithms, the Dense layer is computed through the $Softmax$ activation function. The experiments were carried out by considering all 5 classes present in the NSL-KDD dataset. In Table 11, the outcome shows that for the Simple RNN method, the best classifier attained an accuracy of 84.03%, a F1S of 98.99% and a VAC of 98.93%. This classifier used 150 RNNs units in the hidden layers and it was trained in 109.40 s. With respect to the LSTM method (Table 12), the results show that a model using 150 LSTMs units in 3 layers got a TAC of 85.93%, a F1S of 99.51% and a VAC of 99.47%. In the instance of the

Table 9
LSTM binary classification — NSLKDD.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	S2	98.16%	80.36%	98.38%	63.41
15	3	R2	97.15%	79.74%	97.54%	106.89
30	3	S2	98.80%	80.19%	98.89%	162.06
30	3	R2	98.33%	82.34%	98.56%	157.44
40	3	R2	98.50%	83.23%	98.67%	142.82
40	3	S2	98.51%	80.19%	98.67%	172.34
60	3	R2	98.55%	82.98%	99.28%	134.44
60	3	S2	98.43%	82.94%	98.64%	137.22
90	3	R2	99.28%	84.80%	99.46%	154.51
90	3	S2	99.36%	85.53%	99.35%	148.54
120	3	R2	98.63%	80.38%	98.80%	260.41
120	3	S2	99.39%	84.91%	99.38%	238.46
150	3	R2	99.49%	88.13%	99.58%	225.46
150	3	S2	99.25%	82.26%	99.24%	233.76

Table 10
GRU binary classification — NSLKDD.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	S2	97.82%	83.05%	98.02%	60.94
15	3	R2	97.27%	80.61%	97.54%	61.01
30	3	S2	98.49%	83.16%	98.63%	60.53
30	3	R2	98.35%	83.06%	98.43%	98.51
40	3	R2	99.28%	81.34%	99.30%	145.10
40	3	S2	98.63%	84.56%	98.76%	120.33
60	3	R2	99.11%	80.82%	99.35%	131.40
60	3	S2	98.77%	83.68%	98.89%	142.81
90	3	R2	98.57%	83.03%	98.90%	160.38
90	3	S2	99.37%	85.70%	99.38%	161.00
120	3	R2	99.07%	84.65%	99.10%	211.51
120	3	S2	99.45%	82.26%	99.47%	226.48
150	3	R2	98.29%	82.46%	99.27%	269.26
150	3	S2	99.41%	84.66%	99.37%	276.16

Table 11
SimpleRNN multiclass classification — NSLKDD.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	Sf-5	97.07%	78.57%	97.44%	43.99
30	3	Sf-5	98.28%	79.55%	98.48%	47.88
40	3	Sf-5	98.29%	83.55%	98.49%	56.89
60	3	Sf-5	98.57%	82.66%	98.75%	63.16
90	3	Sf-5	99.23%	82.57%	99.31%	72.16
120	3	Sf-5	98.67%	83.65%	98.83%	97.32
150	3	Sf-5	98.93%	84.03%	98.99%	109.40
180	3	Sf-5	99.42%	81.46%	99.42%	117.77
210	3	Sf-5	99.12%	83.61%	99.22%	123.60

Table 12
LSTM multiclass classification — NSLKDD.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	Sf-5	97.87%	81.90%	98.39%	99.44
30	3	Sf-5	99.28%	82.89%	99.33%	118.23
40	3	Sf-5	98.51%	82.84%	98.70%	121.87
60	3	Sf-5	98.73%	83.12%	98.88%	135.68
90	3	Sf-5	99.29%	84.77%	99.35%	153.16
120	3	Sf-5	99.29%	84.59%	99.33%	216.11
150	3	Sf-5	99.47%	85.93%	99.51%	212.70
180	3	Sf-5	99.49%	83.06%	99.50%	246.73
210	3	Sf-5	99.44%	84.51%	99.44%	395.35

GRU algorithm (Table 13), the best classifier attained a TAC 85.65%, a F1S 99.46% and a training time of 142.07 s. Moreover, in Fig. 8, we performed a comparison of the training times for each model for the multiclass classification task and the trends reveal what was seen in Fig. 7 binary classification training times whereby the Simple RNN is the algorithm that trains faster.

In the second phase of our experimental process, the simulations were conducted on the UNSW-NB15 dataset using its reduced feature

Table 13
GRU multiclass classification — NSLKDD.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	Sf-5	98.28%	80.43%	98.49%	96.19
30	3	Sf-5	99.22%	84.53%	99.24%	95.76
40	3	Sf-5	98.70%	82.52%	98.83%	133.88
60	3	Sf-5	99.35%	83.43%	99.38%	126.48
90	3	Sf-5	99.43%	84.82%	99.43%	160.73
120	3	Sf-5	99.47%	85.49%	99.48%	187.12
150	3	Sf-5	99.42%	85.65%	99.46%	142.07
180	3	Sf-5	99.40%	84.87%	99.45%	184.18
210	3	Sf-5	99.39%	84.76%	99.37%	356.62

Table 14
SimpleRNN binary classification — UNSW-NB15.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	S2	93.51%	82.25%	88.89%	67.03
15	3	R2	93.94%	82.35%	89.83%	64.54
30	3	S2	93.82%	86.10%	89.96%	66.51
30	3	R2	94.01%	82.22%	89.69%	67.12
40	3	S2	94.19%	84.00%	90.37%	66.85
40	3	R2	94.03%	82.27%	89.63%	68.72
60	3	S2	93.93%	83.86%	89.77%	72.48
60	3	R2	93.98%	87.07%	90.03%	68.37
90	3	S2	94.24%	84.35%	90.43%	70.91
90	3	R2	94.12%	84.76%	90.37%	85.07
120	3	S2	94.29%	85.44%	90.76%	95.41
120	3	R2	94.07%	82.85%	89.53%	86.19
150	3	S2	94.52%	83.45%	90.74%	104.42
150	3	R2	94.08%	82.08%	89.77%	110.51

vector as listed in Table 7 (with 17 features). The following were the hyperparameters:

- Binary classification: *loss* = 'binary_crossentropy', *optimiser* = 'sgd' (Stochastic Gradient Descent), initial *learning_rate* = 0.001 (adaptive), *input_shape* = (1, 17)
- Multi-classification: *loss* = 'categorical_crossentropy', *optimiser* = 'adam' (An extension of the Stochastic Gradient Descent), initial *learning_rate* = 0.01 (adaptive), *input_shape* = (1, 17)

As in the first phase, we considered the Simple RNN, LSTM and GRU algorithms. Tables 14–16 depict the results for the binary classification scheme. In Table 14, the most efficient model is a Simple RNN classifier that was configured with 60 units and the *ReLU* activation function in its Dense layer achieved a TAC of 87.07%, a F1S of 90.03% and it produced a training time of 68.37%. The outcome for the LSTM classifier are listed in Table 15 whereby the best model achieved a TAC of 85.08%, a F1S of 90.36% and a VAC of 94.19%. This model was configured with 150 units in the hidden layers and the *ReLU* activation function in its Dense Layer. The results for the GRU classifier are listed in Table 16. In this instance, the most efficient model achieved a TAC of 88.16%, a F1S of 90.45%, a VAC of 94.32% and it produced a training time equal to 196.14 s. The above results demonstrate that in the case of the UNSW-NB15, the usage of the GRU method produces a higher test accuracy in comparison the Simple RNN and the LSTM techniques. However, in terms of training time, as depicted in Fig. 8, the Simple RNN produces a faster training time. Therefore, in instances whereby the computing resources are scarce, using the Simple RNN to implement the proposed framework would be advantageous because of its simplicity.

The outcomes for the multiclass classification schemes using the UNSW-NB15 dataset are presented in Tables 17–19. In the case of the Simple RNN (Table 17), the results demonstrate that the most efficient model achieved a TAC of 74.19%, a F1S of 90.28% and a VAC of 78.67%. This classifier was configured with 150 units in the hidden layers and it produced a training time of 138.43 s. Table 18 shows the results obtained by the LSTM method and the most efficient model

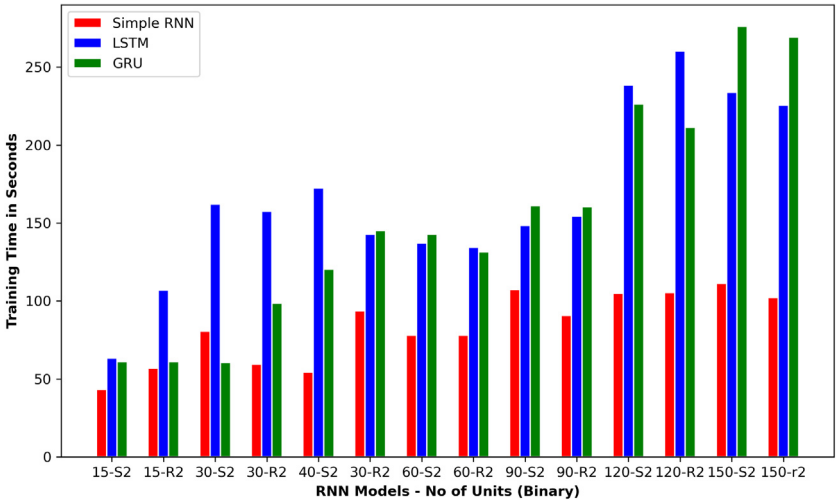

Fig. 7. Training times – Binary classification – NSLKDD.

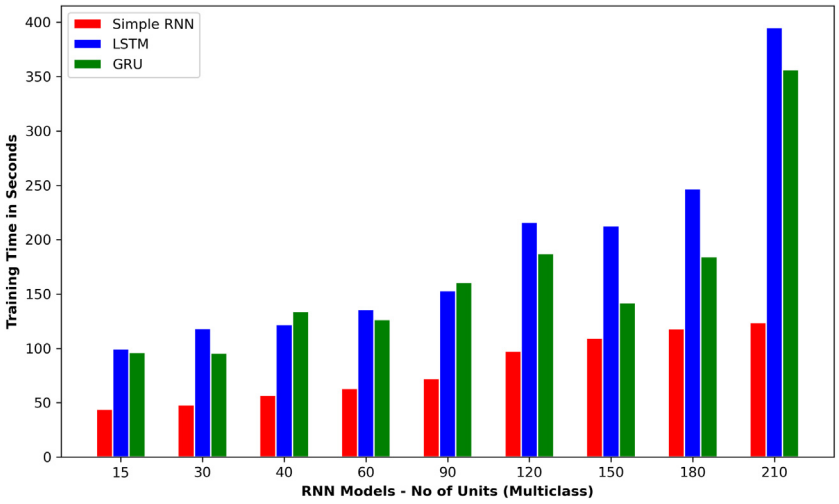

Fig. 8. Training times – Multiclass classification – NSLKDD.

Table 15
LSTM binary classification — UNSW-NB15.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	S2	93.45%	81.04%	88.62%	130.37
15	3	R2	93.73%	81.17%	94.49%	121.39
30	3	S2	94.00%	82.85%	89.72%	121.21
30	3	R2	94.07%	83.91%	89.81%	135.12
40	3	S2	94.08%	83.72%	89.90%	149.57
40	3	R2	94.23%	84.00%	90.28%	145.71
60	3	S2	94.19%	83.50%	90.09%	167.93
60	3	R2	94.28%	84.66%	90.40%	173.25
90	3	S2	94.16%	83.21%	90.30%	189.82
90	3	R2	94.28%	83.93%	90.61%	196.92
120	3	S2	94.23%	84.23%	90.71%	350.63
120	3	R2	94.31%	83.83%	90.48%	297.75
150	3	S2	94.19%	83.80%	90.16%	336.91
150	3	R2	94.19%	85.08%	90.36%	380.46

obtained a TAC of 73.01%, a VAC of 78.68% and it was trained in 405.76 s using 210 units within the hidden layers. In terms of TAC, the performance obtained by the LSTM is superior to the output produced by the best Simple RNN model. Table 19 lists the results obtained by different GRU classifiers and the most efficient one obtained a TAC of 78.40% and VAC of 80.84%. This model was configured using 210 units in the hidden layer. This classifier shows an increase in TAC performance of 4.21% in comparison to the most optimal LSTM method (73.01%) and 5.39% in contrast with the most optimal Simple RNN approach (74.19%). In Fig. 10, we present a comparison of the training times obtained by each model in the instance of the UNSW-NB15 10-way classification task and the trends show that the LSTM is the most complex RNN in terms of how long it trains (see Fig. 9).

In Table 20, we provide a comparison between the proposed methods and the existing binary classification. For the NSL-KDD dataset, the XGBoost-LSTM achieved a TAC of 88.13%. This performance was higher in comparison to other existing methods. For the UNSW-NB15, the XGBoost-GRU attained a TAC of 88.42%. This output was superior in contrast with the methods in [12–14]. In Table 21, a contrast between the methods proposed in this research (PM) and those surveyed in the literature is presented. We listed the relevant techniques and referenced them. The table also mentions the Feature Selection Technique (FST) where applicable. Furthermore, the performance metric used for the comparison analysis is the TAC obtained for the multiclass classification scheme over the UNSW-NB15 and the NSL-KDD datasets. The results show that the LSTM-XGBoost (85.93%) and GRU-XGBoost (85.65%) algorithms performed better than other existing techniques [17,20–22] in the instance of the NSL-KDD datasets. For the UNSW-NB15 dataset, the GRU-XGBoost (78.40%) technique performed optimally in contrast with existing methods [12–14,23].

Additionally, we conducted an analysis on the models that obtained the best multiclass classification accuracy over the UNSW-NB15

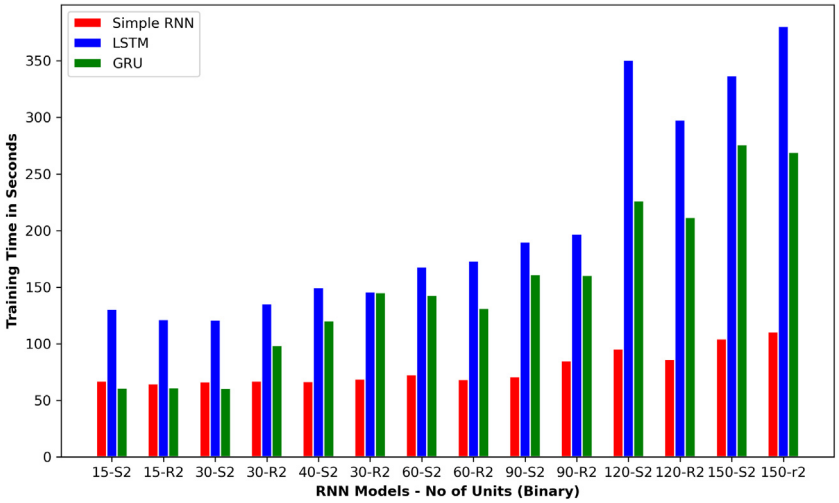

Fig. 9. Training times – Binary classification – UNSW-NB15.

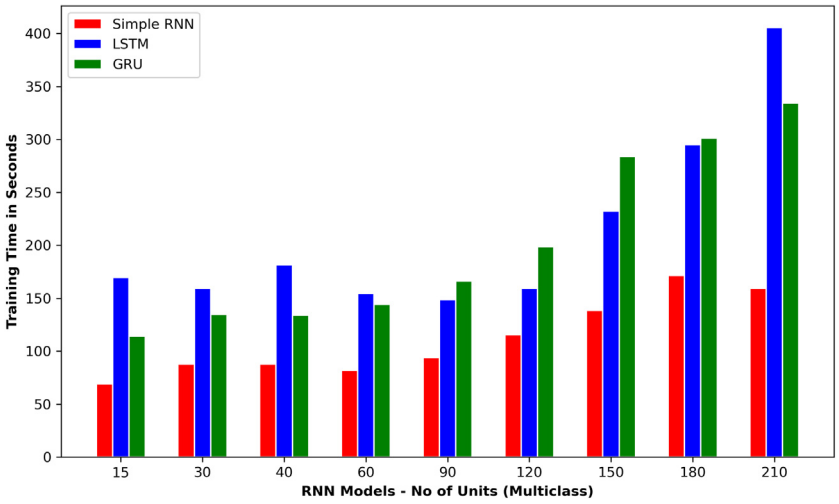

Fig. 10. Training times – Multiclass classification – UNSW-NB15.

(XGBoost-GRU) and the NSL-KDD (XGBoost-LSTM). For each model, we computed a Confusion Matrix (CM) that allowed us to assess the performance of the models on individual classes within the datasets. In Fig. 11, class 0 = *Normal*, class 1 = *R2L*, class 2 =, class 3 = *Probe*, class 4 = *DoS*. In Fig. 12, class 0 = *Normal*, class 1 = *Generic*, class 2 = *Exploits*, class 3 = *Fuzzers*, class 4 = DoS, class 5 = *Reconnaissance*, class 6 = *Analysis*, class 7 = *Backdoor*, class 8 = *Shellcode*, class 9 = *Worms*. The confusion matrix in Fig. 11 demonstrates that the XGBoost-LSTM was efficient in detecting class 0, class 1, 3 and class. Although it was capable of predicting class 1 attacks (which is the second minority class within the NSL-KDD), it was not able to detect class 2 types of attacks. This is because the *U2R* intrusion is the minority class within the NSL-KDD dataset. The confusion matrix in Fig. 12 depicts the analysis of the performance of the XGBoost-GRU on individual classes within the UNSW-NB15 dataset. The results demonstrated that this method performed effectively from class 0–5 and class 7–9.

Table 16
GRU binary classification — UNSW-NB15.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	S2	91.68%	88.42%	87.37%	135.66
15	3	R2	93.92%	84.12%	89.61%	134.97
30	3	S2	94.11%	85.42%	90.00%	132.04
30	3	R2	94.03%	83.07%	89.78%	136.77
40	3	S2	93.90%	86.89%	90.11%	176.96
40	3	R2	94.14%	83.13%	90.25%	143.86
60	3	S2	94.04%	84.09%	89.80%	170.51
60	3	R2	94.04%	82.57%	89.69%	160.43
90	3	S2	94.32%	88.16%	90.45%	196.14
90	3	R2	94.20%	84.32%	90.23%	183.29
120	3	S2	94.15%	84.52%	90.49%	212.12
120	3	R2	94.26%	84.87%	90.59%	204.64
150	3	S2	94.18%	82.35%	90.13%	297.71
150	3	R2	94.21%	86.83%	90.44%	353.70

9. Conclusion

This study presented the implementation of an IDS framework using different types of RNNs techniques coupled with the XGBoost-based feature selection method. The NSL-KDD and UNSW-NB15 datasets were utilized to assess the performance of the classifiers. The application of the XGBoost-FST over the NSL-KDD and the UNSW-NB15 have resulted in the extraction of 22 and 17 most relevant attributes, respectively. Moreover, a comparison analysis of the training times of each model was conducted. The RNNs that were used during the experimental process are the LSTM, the Simple RNN and the GRU. Furthermore, the framework proposed in this research was compared to other existing methods that were surveyed in the literature. The results demonstrated that in the case of the binary classification scheme, the XGBoost-LSTM

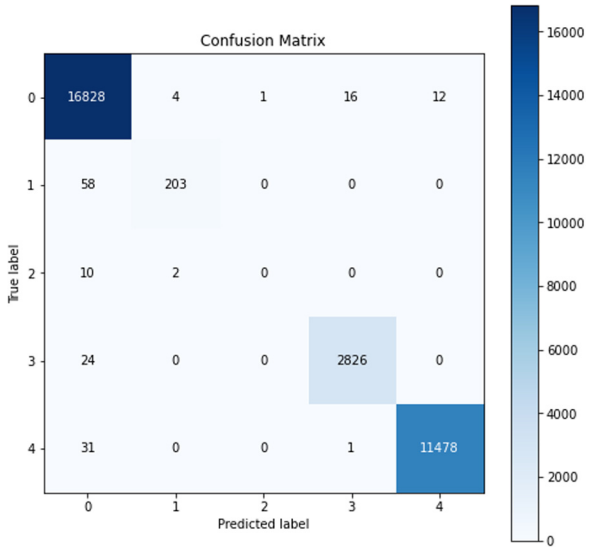

Fig. 11. XGBoost-LSTM – Confusion matrix – NSL-KDD.

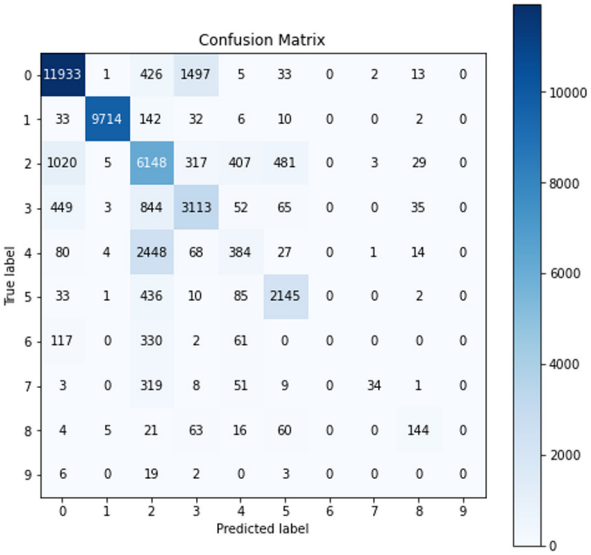

Fig. 12. XGBoost-LSTM – Confusion matrix – NSL-KDD.

Table 17
SimpleRNN multiclass classification — UNSW-NB15.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	Sf-5	76.20%	66.66%	88.64%	68.94
30	3	Sf-5	79.56%	69.87%	89.80%	87.70
40	3	Sf-5	79.49%	69.77%	90.20%	87.68
60	3	Sf-5	79.76%	69.47%	90.15%	81.87
90	3	Sf-5	78.82%	72.17%	90.34%	93.74
120	3	Sf-5	79.09%	71.21%	90.63%	115.34
150	3	Sf-5	78.67%	74.19%	90.26%	138.43
180	3	Sf-5	78.37%	70.00%	90.11%	171.47
210	3	Sf-5	80.41%	72.42%	90.28%	159.25

Table 18
LSTM multiclass classification — UNSW-NB15.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	Sf-5	76.96%	70.12%	89.47%	169.69
30	3	Sf-5	77.05%	70.45%	88.76%	159.29
40	3	Sf-5	78.76%	72.25%	89.53%	181.38
60	3	Sf-5	78.29%	73.79%	90.16%	154.68
90	3	Sf-5	78.23%	69.72%	90.20%	148.60
120	3	Sf-5	78.88%	72.26%	90.42%	159.40
150	3	Sf-5	78.81%	71.24%	90.54%	232.45
180	3	Sf-5	78.75%	70.56%	90.43%	294.91
210	3	Sf-5	78.68%	73.01%	90.35%	405.76

Table 19
GRU multiclass classification — UNSW-NB15.

NU	HS	Dense	VAC	TAC	F1S	Ts
15	3	Sf-5	77.71%	70.98%	87.64%	114.10
30	3	Sf-5	79.06%	70.94%	89.82%	134.86
40	3	Sf-5	77.48%	72.52%	89.90%	133.87
60	3	Sf-5	79.72%	71.48%	90.14%	144.25
90	3	Sf-5	78.30%	69.05%	90.07%	166.27
120	3	Sf-5	79.50%	69.38%	90.19%	198.55
150	3	Sf-5	80.36%	72.18%	90.28%	283.64
180	3	Sf-5	78.83%	70.67%	90.99%	301.18
210	3	Sf-5	80.84%	78.40%	80.84%	334.12

Table 20
Comparison with other methods (Binary classification).

ML Method	Dataset	FST	TAC
MCA-LSTM [12]	UNSW-NB15	IG	88.11%
RNN [13]	UNSW-NB15	–	83.28%
FFDNN [14]	UNSW-NB15	ExtraTrees	87.10%
Simple RNN [PM]	UNSW-NB15	XGBoost	87.07%
LSTM [PM]	UNSW-NB15	XGBoost	85.08%
GRU [PM]	UNSW-NB15	XGBoost	88.42%
MCA-LSTM [12]	NSL-KDD	IG	80.52%
CNN-LSTM [17]	NSL-KDD	–	74.77%
Simple RNN [PM]	NSL-KDD	XGBoost	83.70%
LSTM [PM]	NSL-KDD	XGBoost	88.13%
GRU [PM]	NSL-KDD	XGBoost	84.66%

Table 21
Comparison with other methods (Multiclass classification).

ML Method	Dataset	FST	TAC
MCA-LSTM [12]	UNSW-NB15	IG	77.74%
RNN [13]	UNSW-NB15	–	81.29%
FFDNN [14]	UNSW-NB15	ExtraTrees	77.16%
ANN-MLP [24]	UNSW-NB15	GR	76.96%
Simple RNN [PM]	UNSW-NB15	XGBoost	74.19%
LSTM [PM]	UNSW-NB15	XGBoost	73.01%
GRU [PM]	UNSW-NB15	XGBoost	78.40%
MCA-LSTM [12]	NSL-KDD	IG	82.15%
CNN-LSTM [17]	NSL-KDD	–	68.78%
GRU [20]	NSL-KDD	–	82.87%
Sem-LSTM [19]	NSL-KDD	–	82.21%
LSTM [20]	NSL-KDD	–	82.78%
LSTM [21]	NSL-KDD	–	82.00%
DNN [22]	NSL-KDD	Empirical	75.75%
Simple RNN [PM]	NSL-KDD	XGBoost	84.03%
LSTM [PM]	NSL-KDD	XGBoost	85.93%
GRU [PM]	NSL-KDD	XGBoost	85.65%

achieved the best performance with a TAC of 88.13% (NSL-KDD) and the XGBoost-Simple-RNN obtained a TAC of 87.07% (UNSW-NB15). For the multiclass configuration, the XGBoost-LSTM obtained a TAC of 86.93% (NSL-KDD) and the XGBoost-GRU got a TAC of 78.40%. The results obtained by our framework were superior to those attained by existing methods. Additionally, the training time comparison analysis of each model revealed that in all instances, the XGBoost-Simple-RNN produced a faster training time. Therefore, in an environment whereby computing resources are limited, the Simple-RNN would be the optimal choice. In future studies, the aim is to investigate the performance of the proposed framework on individual classes contained within the studied datasets and to increase the models performance on minority classes. Additionally, a study of hybrid methods (combination of different RNN models) will be implemented.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: A systematic study of machine learning and deep learning approaches, Trans. Emerg. Telecommun. Technol. 32 (1) (2021) 000000.

[2] R. Vinayakumar, M. Alazab, K.P. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep learning approach for intelligent intrusion detection system, IEEE Access 7 (2019) 41525–41550.

[3] Z. Chkirbene, A. Erbad, R. Hamila, A. Mohamed, M. Guizani, M. Hamdi, TIDCS: A dynamic intrusion detection and classification system based feature selection, IEEE Access 8 (2020).

[4] E. Alpaydin, Introduction to Machine Learning, MIT Press, 2020.

[5] M. Botvinick, S. Ritter, J.X. Wang, Z. Kurth-Nelson, C. Blundell, D. Hassabis, Reinforcement learning, fast and slow, Trends Cogn. Sci. 23 (5) (2019) 408–422.

[6] S. Raschka, V. Mirjalili, Python Machine Learning, Packt Publishing Ltd, 2017.

[7] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[8] Y. Wang, W. Liao, Y. Chang, Gated recurrent unit network-based short-term photovoltaic forecasting, Energies 11 (8) (2018) 2163.

[9] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM Sigkdd Int Conf. on KDD, 2016, pp. 785–794.

[10] G. Meena, R.R. Choudhary, A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA, in: Int. Conf. on Comput. Commun. Electron., IEEE, 2017, pp. 553–558.

[11] M.S. Al-Daweri, K.A. Zainol Ariffin, S. Abdullah, An analysis of the KDD99 and UNSW-NB15 datasets for the intrusion detection system, Symmetry 12 (10) 1666.

[12] R.H. Dong, X.Y. Li, Q.Y. Zhang, H. Yuan, Network intrusion detection model based on multivariate correlation analysis–long short-time memory network, IET Inf. Secur. 14 (2) (2019) 166–174.

[13] C. Yin, Y. Zhu, J. Fei, X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access 5 (2017) 21954–21961.

[14] S.M. Kasongo, Y. Sun, A deep learning method with wrapper based feature extraction for wireless intrusion detection system, Comput. Secur. 92 (2020) 101752.

[15] C. Kolias, G. Kambourakis, A. Stavrou, S. Gritzalis, Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset, IEEE Commun. Surv. Tutor. 18 (1) (2015) 184–208.

[16] Ali H. Mirza, S. Cosan, Computer network intrusion detection using sequential LSTM neural networks autoencoders, in: 26th Signal Process. Commun. Appli. Conf., SIU, 2018, pp. 1–4.

[17] C.M. Hsu, H.Y. Hsieh, S.W. Prakosa, M.Z. Azhari, J.S. Leu, Using long-short-term memory based convolutional neural networks for network intrusion detection, in: Int. Wir. Internet Conf. Springer Cham. Oct., 2018, pp. 86–94.

[18] S. Miao, J.Z. Wang, R. Liao, Convolutional Neural Networks for Robust and Real-Time 2-D/3-D Registration, Deep Learning for Medical Image Analysis, Academic Press, 2017, pp. 271–296.

[19] Z. Li, Z.A. Qin, A semantic parsing based LSTM model for intrusion detection, in: Int. Conf. Neural Inf. Process., Springer, Cham, 2018, pp. 600–609.

[20] Z. Li, A.L.G. Rios, G. Xu, L. Trajković, Machine learning techniques for classifying network anomalies and intrusions, in: IEEE Int. Symposium on Circuits and Sys., ISCAS, 2019, pp. 1–5.

[21] S. Wang, C. Xia, T. Wang, A novel intrusion detector based on deep learning hybrid methods, in: 5th Int. Conf. Big Data Sec. on Cloud, IEEE Int. Conf. on High Performance and Smart Computing (HPSC) and IEEE Int. Conf. Int. Data and Security, 2019, pp. 300–305.

[22] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: Int. Conf. Wireless Netw. Mobile Commun., WINCOM, IEEE, 2016, pp. 258–263.

[23] C. Khammassi, S. Krichen, A GA-LR wrapper approach for feature selection in network intrusion detection, Comput. Secur. 70 (2017) 255–277.

[24] J.O. Mebawondu, O.D. Alowolodu, J.O. Mebawondu, A.O. Adetunmbi, Network intrusion detection system using supervised learning paradigm, Scientific African 9 (2020) e00497.

[25] E. Heidari, M.A. Sobati, S. Movahedirad, Accurate prediction of nanofluid viscosity using a multilayer perceptron artificial neural network (MLP-ANN), Chemometr. Intell. Lab. Syst. 15 (155) (2016) 73–85.

[26] J. Kim, H. Kim, An effective intrusion detection classifier using long short-term memory with gradient descent optimization, in: 2017 Int. Conf. on Platform Tech. Serv., PlatCon, IEEE, 2017, pp. 1–6.

[27] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117.

[28] J.A. Bullinaria, Recurrent neural networks, in: Neural Computation: Lecture, Vol. 12, 2013.

[29] F. Meng, Y. Fu, F. Lou, Z. Chen, An effective network attack detection method based on kernel PCA and LSTM-RNN, in: Int. Conf. Comp. Sys. Elect. Ctrl, ICCSEC IEEE, 2017, pp. 568–572.

[30] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep recurrent neural network for intrusion detection in SDN-based networks‘, in: IEEE Conf. on Ntw. Softwarization and Workshops, NetSoft, 2018, pp. 202–206.

[31] The NSL-KDD : NSL KDD dataset [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html.

[32] G.C. Cawley, N.L. Talbot, On over-fitting in model selection and subsequent selection bias in performance evaluation, J. Mach. Learn. Res. 11 (2010) 2079–2107.

[33] N. Moustafa, J. Slay, UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), in: Military Comm. Inf. Sys. Conf., MilCIS, IEEE, 2015, pp. 1–6.

[34] A.F. Agarap, Deep learning using rectified linear units (ReLU), 2018, arXiv preprint arXiv:1803.08375.

[35] D. Zhu, S. Lu, M. Wang, J. Lin, Z. Wang, Efficient precision-adjustable architecture for softmax function in deep learning, IEEE Trans. Circuits Syst. II (2020).

[36] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, Y. Bengio, Batch normalized recurrent neural networks, in: Int. Conf. on Acoustics, Speech and Signal Process., ICASSP, IEEE, 2016, pp. 2657–2661.

[37] Z. Liu, A method of SVM with normalization in intrusion detection, Procedia Environ. Sci. 11 (2011) 256–262.

[38] M. Gumus, M.S. Kiran, Crude oil price forecasting using XGBoost, in: Int. Conf. Comp. Sci. Eng., IEEE, 2017, pp. 1100–1103.

[39] X. Ren, H. Guo, S. Li, S. Wang, J. Li, A novel image classification method with CNN-XGBoost model, in: Int. Workshop on Digital Watermarking, Springer, Cham, 2017, pp. 378–390.

[40] J. Friedman, Greedy function approximation, a gradient boosting machine, Ann. Statist. 29 (5) (2001) 1189–1232.

[41] A. Ogunleye, Q.G. Wang, XGBoost model for chronic kidney disease diagnosis, IEEE/ACM Trans. Comput. Biol. Bioinform. 17 (6) (2019) 2131–2140.

[42] Scikit-learn, machine learning in Python, 2020, [Online]. Available: https://scikit-learn.org/stable/. (Accessed on: 03 September 2020).

[43] Keras, deep learning library, 2020, [Online]. Available: https://keras.io/. (Accessed on: 3 September 2020).