

RESEARCH ARTICLE

Comparative Analysis of Random Forest and Artificial Neural Network Models for Network Traffic Anomaly Detection on the NSL-KDD Dataset

Cennet Büşra HAKAY ^a, , Sevdanur GENÇ ^{b*}

^a Department of Cyber Security and Cryptography, Yıldız Technical University, 34220, Istanbul, Türkiye

^b Department of Computer Engineering, Balıkesir University, 10145, Balıkesir, Türkiye

* Corresponding author: sevdanur.genc@balikesir.edu.tr · ORCID: 0000-0003-4774-9265

ARTICLE INFO

Article history

Received 21 June 2026

Revised 28 June 2026

Accepted 29 June 2026

Available online 29 June 2026

Keywords

Network intrusion detection

Anomaly detection

NSL-KDD

Random forest

Artificial neural networks

False alarm rate

Abstract

The volume and sophistication of network attacks have grown faster than the signature databases on which conventional intrusion detection systems depend, and behaviour-based anomaly detection has become the principal alternative. This paper reports a controlled comparison of a random forest (RF) classifier and a multilayer perceptron artificial neural network (ANN) for binary anomaly detection on the NSL-KDD benchmark, which removes the duplicate records and the unrealistic class distribution of its KDD Cup 1999 predecessor. Both models are trained under a single preprocessing pipeline consisting of label standardisation, one-hot encoding of the three nominal attributes, a logarithmic transformation of the heavy-tailed volume attributes, min-max scaling to the unit interval, and synthetic minority over-sampling applied to the training partition only. Evaluation on the held-out KDDTest+ partition shows that the two architectures separate along a false-alarm/detection trade-off rather than along overall accuracy. The random forest reaches 76.55% accuracy with 267 false positives and an area under the receiver operating characteristic curve of 0.9609, whereas the neural network reaches 79.85% accuracy and detects 1,221 more attacks, at the cost of 742 false positives and a lower area under the curve of 0.8797. Both models exceed 99% accuracy on the training partition, so the drop of roughly twenty percentage points on the test partition is attributed to attack families that are absent from the training data rather than to a deficiency of either architecture. The serialised models are additionally exposed through a packet-capturing web service that demonstrates the deployment path end to end, with the limits of the present packet-to-feature mapping stated explicitly.

1. Introduction

The rapid transformation of information technology and the exponential growth in the number of Internet-connected devices have made cyberspace a critical infrastructure of modern society. That expansion has been accompanied by a comparable growth in the volume and the sophistication of cyber threats. Zero-day exploits and polymorphic malware in particular have begun to defeat conventional firewalls and signature-based intrusion detection systems (IDS) [1]. A signature-based IDS operates by matching traffic against attack patterns already recorded in its

database, so it is structurally unable to recognise a vector it has never seen or one that has changed shape. Closing this gap requires detection systems that learn from data and identify deviations from normal behaviour — anomalies — through behavioural analysis of network traffic [2].

Anomaly detection in network traffic is among the most successful application areas of machine learning (ML) and deep learning (DL). Such systems model normal network behaviour from large volumes of historical traffic and flag activity that does not conform to that model as a potential threat. The effectiveness of supervised algorithms such as support vector machines (SVM), k-nearest neighbours (k-NN) and decision trees has been investigated extensively in this setting [3], [4]. A large share of that work, however, rests on the KDD Cup 1999 dataset [5], which contains a substantial proportion of duplicate records. Tavallaee et al. [6] showed that these structural defects lead algorithms to memorise the data rather than learn from it, and therefore to report misleadingly high success rates.

The NSL-KDD dataset [7] was constructed to remove those defects: duplicate records are eliminated and the class distribution is rebalanced, which makes it a more reliable benchmark for the development and comparison of intrusion detection systems [8]. Studies examining the performance of random forest (RF) and artificial neural network (ANN) models on NSL-KDD already exist [9], [10], but most of them report accuracy alone. Two quantities that determine whether a detector is usable in production are usually left in the background: the false positive rate, which governs how much noise the system generates for a security team, and the cost of integrating a trained model into a live pipeline.

This study addresses network anomaly detection with an RF classifier and an ANN based on the multilayer perceptron (MLP), both trained on NSL-KDD under a single, shared preprocessing pipeline. The aim is not to obtain a high headline accuracy but to characterise the operational trade-off between the two architectures and to show that the resulting models can be deployed. To that end the models are analysed through per-class precision, recall and F1-score, through confusion matrices and through the receiver operating characteristic (ROC) curve, and are then integrated into a web service capable of classifying captured packets in real time.

1.1. Contributions

The contributions of this work are the following.

- (i) A like-for-like comparison of random forest and artificial neural network models on NSL-KDD in which the preprocessing pipeline, the training partition and the evaluation protocol are held fixed, so that the observed differences are attributable to the classifier rather than to the data preparation.
- (ii) An analysis of the detection/false-alarm trade-off that separates the two models, reported through confusion matrices and the area under the ROC curve rather than accuracy alone, together with an account of the generalisation gap between the KDDTrain+ and KDDTest+ partitions.
- (iii) A real-time IDS prototype in which the serialised models are served through a Flask application that captures live packets and displays the predictions of both models side by side. The limits of the present packet-to-feature mapping are stated explicitly in Section 5.

1.2. Organisation

Section 2 reviews related work. Section 3 describes the dataset, the preprocessing pipeline and the two model architectures. Section 4 reports the experimental results and the comparative analysis. Section 5 describes the design of the real-time detection system, and Section 6 discusses the limitations of the study and directions for further work.

2. Related Work

Research on cyber security and network anomaly detection has concentrated on two axes: improving the quality of the benchmark data, and maximising classification performance. On the first axis, the high rate of duplicate records and the statistical imbalance of the KDD Cup 1999 dataset were shown to distort measured model performance, and NSL-KDD, in which those defects are corrected, has since been treated as the reference benchmark for developing and comparing intrusion detection systems. On the second axis, comparative studies have examined tree ensembles,

kernel methods and neural architectures on the same data; Revathi and Malathi [11], for instance, report that decision-tree-based methods give the highest detection accuracy on NSL-KDD. Table 1 summarises a representative selection of this literature.

Table 1. Representative studies on machine learning and deep learning methods for network anomaly detection.

Study	Dataset	Method	Principal finding
Tavallaee et al. [6]	KDD Cup 1999, NSL-KDD	Statistical analysis	Identified a duplicate-record rate of approximately 78% in KDD Cup 1999 and proposed NSL-KDD as a corrected benchmark.
Revathi and Malathi [11]	NSL-KDD	Random forest, J48, CART	Decision-tree-based methods gave the highest detection accuracy; emphasised the value of NSL-KDD for detecting modern attack types.
Dhanabal and Shantharajah [8]	NSL-KDD	J48, SVM, naïve Bayes	Compared classifiers and showed that reducing the dimensionality of the traffic data through feature selection improves performance.
Ingre and Yadav [10]	NSL-KDD	Artificial neural network	Reported detection rates of 81.2% for binary classification and 79.9% for five-class classification of attack types.
Belavagi and Muniyal [9]	NSL-KDD	Random forest, SVM, Gaussian naïve Bayes, logistic regression	Random forest gave higher accuracy and more balanced results than the alternative classifiers.
Ahmad et al. [3]	NSL-KDD	SVM, random forest, extreme learning machine	The extreme learning machine outperformed both support vector machines and random forest on the full dataset.

Taken together, these comparative results indicate that ensemble methods built on decision trees remain stable even when the class distribution is skewed, and that they generally produce lower false alarm rates than support vector machines or naïve Bayes. Neural architectures, by contrast, model the non-linear relationships in network traffic with greater capacity but at a higher computational cost. A further point of agreement is that filtering the most discriminative attributes, rather than feeding the full high-dimensional vector to the classifier, improves both detection accuracy and processing speed. The present study takes this consensus as its starting point and asks what remains once the preprocessing pipeline is held fixed and only the classifier is varied.

3. Materials and Methods

3.1. Dataset

The validity of intrusion detection research depends directly on how realistic and how balanced the underlying dataset is. Two datasets that dominated earlier work — DARPA 1998/99 [12] and KDD Cup 1999 [5] — carry well-documented methodological limitations. KDD Cup 1999 is criticised for a duplicate-record rate of approximately 78% and for a severely skewed class distribution in which denial-of-service (DoS) traffic is heavily over-represented while the user-to-root (U2R) and remote-to-local (R2L) categories are almost absent. These defects allow an algorithm to memorise the data rather than learn the structure of the traffic. The DARPA data, generated synthetically in a military laboratory environment, likewise fail to reflect modern traffic behaviour, current attack types and present-day protocols.

NSL-KDD [7] was constructed to address these shortcomings and is used here as the primary data source. It is an optimised version of KDD Cup 1999 in which duplicate records are removed, the training and test partitions are more evenly distributed, and records are sampled according to their difficulty level. These properties stabilise the learning process and have made NSL-KDD one of the most widely used reference benchmarks in the current IDS literature [8].

Each traffic record consists of 41 independent attributes and one target label. The files also carry a difficulty-level column, which is not a property of the traffic and was dropped before training. Three attributes — *protocol_type*, *service* and *flag* — are nominal; the remaining 38 are numeric. The labels distinguish normal traffic from attack traffic, and attack traffic is further divided into four categories: DoS, Probe (reconnaissance), U2R and R2L. To simplify the classification task and to evaluate a binary formulation, the labels were re-encoded here as Normal (0) and Attack (1). Under this encoding KDDTrain+ contains 125,973 records (67,343 normal and 58,630 attack) and KDDTest+ contains 22,544 records (9,711 normal and 12,833 attack). Figure 1 shows a sample of the raw records.

Figure 1. Preview of the NSL-KDD dataset. The excerpt shows the structure of the categorical and numeric attributes together with the label format used for the normal and attack classes.

The performance of a machine learning model depends heavily on the preprocessing applied to its input. The pipeline described in the following subsections was applied identically to both models except where stated.

A structural inspection of NSL-KDD found no missing values, but formatting inconsistencies were present in the text-based attributes and in the class labels. A standardisation step was therefore applied: variant spellings of the label values were unified, and redundant whitespace and format mismatches were removed. Beyond preserving data integrity, this step ensures that the encoding and modelling stages that follow are not disrupted by parsing errors.

The *protocol_type* (tcp, udp, icmp), *service* (70 distinct values such as http, private and ftp) and *flag* (11 connection states such as SF and S0) attributes are nominal and must be converted to a numeric representation before either model can process them. Because no ordinal relation holds between their categories, one-hot encoding [13] was used so that the models would not infer a spurious ordering such as tcp > udp. Each category becomes a binary column that takes the value 1 for the corresponding class and 0 otherwise.

4

3.2.3. Outlier handling and logarithmic transformation

Network traffic attributes such as *src_bytes* and *dst_bytes* span very wide ranges and are strongly right-skewed by nature. Volumetric attacks such as DoS are statistically outliers, yet they are precisely the records that matter for intrusion detection. Outliers were therefore not removed. Instead, a logarithmic transformation [14] was applied to the seven heavy-tailed attributes *duration*, *src_bytes*, *dst_bytes*, *count*, *srv_count*, *dst_host_count* and *dst_host_srv_count* in order to compress the scale of the data and to keep the gradients of the neural network stable during training:

$$x' = \ln(1 + x) \quad (1)$$

The random forest is invariant to monotone transformations of individual attributes and is robust to outliers by construction, so this transformation was applied in the neural network pipeline only.

3.2.4. Normalisation

The numeric attributes are distributed over widely different ranges. To prevent the gradient-based learning of the neural network from being slowed by these scale differences, and to prevent the model from becoming biased towards large-valued attributes, all numeric attributes were rescaled to the unit interval by min–max normalisation [15]:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (2)$$

The minimum and maximum of each attribute were computed on the training partition and applied unchanged to the test partition, so that no information from the test data enters the transformation. Figure 2 shows the categorical attributes in their raw form before encoding; after one-hot encoding and the two scaling steps the dataset is reduced to a single numeric matrix of 122 columns without any loss of information.

```
protocol_type ['icmp' 'tcp' 'udp']
service ['IRC' 'X11' 'Z39_50' 'aol' 'auth' 'bgp' 'courier' 'csnet_ns' 'ctf'
'daytime' 'discard' 'domain' 'domain_u' 'echo' 'eco_i' 'ecr_i' 'efs'
'exec' 'finger' 'ftp' 'ftp_data' 'gopher' 'hostnames' 'http' 'http_443'
'http_8001' 'imap4' 'iso_tsap' 'klogin' 'kshell' 'ldap' 'link' 'login'
'mtp' 'name' 'netbios_dgm' 'netbios_ns' 'netbios_ssn' 'netstat' 'nntp'
'nntp' 'ntp_u' 'other' 'pm_dump' 'pop_2' 'pop_3' 'printer' 'private'
'red_i' 'remote_job' 'rje' 'shell' 'smtp' 'sql_net' 'ssh' 'sunrpc'
'supdup' 'systat' 'telnet' 'tftp_u' 'tim_i' 'time' 'urh_i' 'urp_i' 'uucp'
'uucp_path' 'vmnet' 'whois']
flag ['OTH' 'REJ' 'RSTO' 'RSTO50' 'RSTR' 'S0' 'S1' 'S2' 'S3' 'SF' 'SH']
```

Figure 2. The categorical attributes *protocol_type*, *service* and *flag* in their original, unencoded form.

3.2.5. Feature-importance and correlation analysis

To identify which attributes carry the most discriminative information, the random forest feature-importance scores and a correlation heat map of the encoded attributes were examined together. The analysis isolated a small group of attributes that dominate the split decisions and several groups of one-hot columns that are close to collinear. The models reported in Section 4 were nevertheless trained on the complete 122-dimensional encoded vector; the analysis is therefore reported here as a diagnostic of the feature space rather than as a dimensionality-reduction step.

3.2.6. Class imbalance and synthetic over-sampling

The class distribution of NSL-KDD is dominated by the normal and DoS categories, whereas U2R and R2L are represented by very few records. Such imbalance leads a classifier to concentrate on the majority class and to overlook rare but critical attack types. The synthetic minority over-sampling technique (SMOTE) [16] was applied to counteract this. SMOTE generates new records by interpolating between existing minority-class points along the edges of their k-nearest-neighbour graph. To preserve methodological consistency and to prevent data leakage, SMOTE was applied only after the training and test partitions had been separated, and only to the training partition.

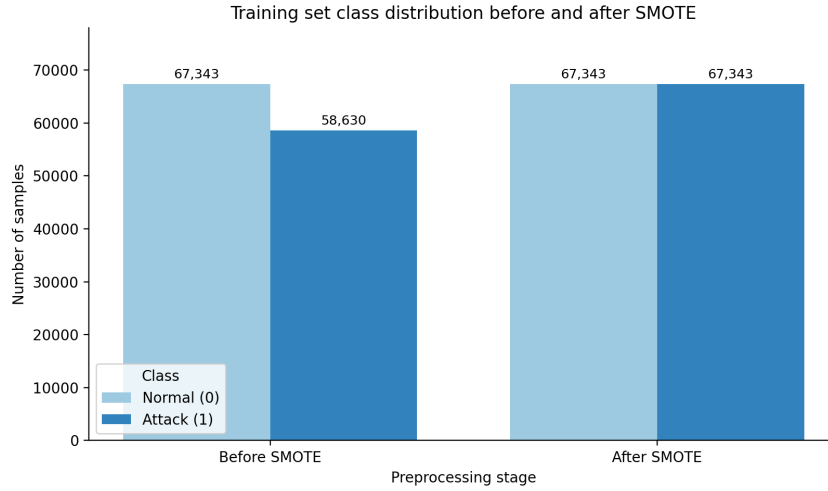


Figure 3. Class distribution of the training partition used for the SMOTE operation.

As Figure 3 shows, the normal class held 67,343 records against 58,630 for the attack class before the operation. Applying SMOTE with *sampling_strategy* = ‘not majority’ raised the minority class synthetically until both classes contained 67,343 records, giving a fully balanced training partition of 134,686 records. This prevents the models from making decisions biased towards the majority class and increases their sensitivity to rare attack types.

3.3. Random forest model and hyperparameter optimisation

Random forest is an ensemble method built from decision trees [17]. It uses bootstrap sampling to derive different training subsets from the original data and trains each tree on a randomly selected subspace of the attributes. The prediction of the forest for a sample is determined by majority voting over the predictions of the individual trees. Writing $h(x)$ for the prediction of the i -th decision tree and n for the number of trees, this mechanism is given by (3):

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), \dots, h_n(x)\} \quad (3)$$

Node splitting used the Gini impurity criterion, which measures the probability that a randomly chosen element of a set is misclassified. With p denoting the proportion of class k in the node and K the number of classes, the Gini coefficient is computed as in (4):

$$G = \sum_{k=1}^K p_k(1 - p_k) \quad (4)$$

Hyperparameter optimisation was carried out with RandomizedSearchCV, sampling ten candidate configurations from a grid over the number of trees ($n_estimators \in \{100, 200\}$), the maximum tree depth ($max_depth \in \{10, 20, \text{None}\}$), the minimum number of samples required to split an internal node ($min_samples_split \in \{2, 5\}$), the minimum number of samples required at a leaf ($min_samples_leaf \in \{1, 2\}$) and the use of bootstrap resampling ($bootstrap \in \{\text{True}, \text{False}\}$). The splitting criterion was fixed to Gini impurity, candidates were scored by the F1-score of the attack class under three-fold cross-validation, and *class_weight* was set to *balanced* so that any residual asymmetry between the classes was penalised in addition to the over-sampling applied to the training partition. The selected configuration used 100 trees with no limit on tree depth ($max_depth = \text{None}$), $min_samples_split = 2$, $min_samples_leaf = 1$ and $bootstrap = \text{False}$; in the selected model the diversity among trees therefore derives from the random feature subspace alone rather than from bootstrap resampling. All remaining parameters were left at their scikit-learn defaults and the random seed was fixed to 42.

3.4. Neural network architecture and training strategy

A multilayer perceptron was used to model the non-linear relationships in the traffic data. The topology consists of an input layer matching the 122-dimensional encoded feature vector, three hidden layers of 128, 64 and 32 neurons, and a single-neuron output layer that produces the binary decision. The hidden layers use the rectified linear unit (ReLU) activation given in (5), which zeroes negative activations, encourages sparse representations and limits the vanishing-gradient problem:

$$f(x) = \max(0, x) \quad (5)$$

The output layer applies a sigmoid activation in order to map the score produced by the network to a probability in the interval $[0, 1]$.

Weights were updated with the Adam optimiser [18], which combines adaptive per-parameter step sizes with momentum. With the bias-corrected first and second moment estimates at step t denoted by a circumflex, α the base step size and ε the damping constant, the update rule is given by (6):

$$\theta_{t+1} = \theta_t - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \quad (6)$$

Two mechanisms were used to control overfitting. Dropout at a rate of 0.3 was applied after each hidden layer to reduce co-adaptation between neurons, and early stopping monitored the validation loss with a patience of ten epochs and restored the weights of the best epoch. Batch normalisation was applied in each hidden layer and the loss function was binary cross-entropy. The network was trained with the Adam optimiser at its default settings (learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$) and a batch size of 64, for a maximum of 100 epochs, with 20% of the balanced training partition held out for validation. The lowest validation loss, 0.0046, was reached at epoch 49, and the weights of that epoch were restored for all results reported below. Table 2 summarises the layer structure of the model.

Table 2. Architecture of the artificial neural network and layer properties.

Layer	Size (neurons)	Activation and regularisation
Input layer	122 (after one-hot encoding)	—
Hidden layer 1	128	ReLU + batch normalisation + dropout (0.3)
Hidden layer 2	64	ReLU + batch normalisation + dropout (0.3)
Hidden layer 3	32	ReLU + batch normalisation + dropout (0.3)
Output layer	1	Sigmoid

3.5. Evaluation strategy and performance metrics

Model performance was analysed with accuracy, precision, recall, F1-score, the area under the ROC curve (ROC-AUC) and the confusion matrix. Accuracy alone is known to be a misleading indicator when the class distribution is skewed, so the F1-score and ROC-AUC were adopted as the primary indicators of this study. Precision, recall and F1-score are reported for each class separately, and the confusion matrix is reported in absolute counts, so that the two error types — false alarms and missed attacks — can be weighed against each other rather than collapsed into a single figure.

4. Experimental Results

Two machine learning approaches were applied once preprocessing was complete: a random forest (RF) and a multilayer perceptron artificial neural network (ANN). Using two structurally different models makes it possible to compare a classical ensemble method with a deep learning method on the same complex behavioural patterns.

Unless stated otherwise, all results below are reported on the held-out KDDTest+ partition, which deliberately contains attack families that do not appear in the training data.

4.1. Random forest

Table 3 reports the overall performance of the model on the KDDTest+ partition, where it reaches an accuracy of 76.55%. Examined per class, the model attains a recall of 0.97 on normal traffic (class 0) but only 0.61 on attack traffic (class 1). The precision of 0.97 obtained for the attack class shows that when the model does label a record as an attack, that decision is correct in the overwhelming majority of cases. F1-scores of 0.78 and 0.75 for the normal and attack classes indicate that the model preserves a degree of balance between the two, while the drop on a test set containing unseen attack types points to a generalisation capability that can be improved.

Table 3. Classification report of the random forest model on the KDDTest+ partition. The random forest was evaluated on 22,543 of the 22,544 records of KDDTest+, one record having been lost when the file was parsed; the effect on the reported rates is below 0.01 percentage points.

Class	Precision	Recall	F1-score	Support
Normal (0)	0.65	0.97	0.78	9,711
Attack (1)	0.97	0.61	0.75	12,832
Accuracy	—	—	0.7655	22,543
Macro average	0.81	0.79	0.76	22,543
Weighted average	0.83	0.77	0.76	22,543
ROC-AUC	—	—	0.9609	—

Figure 4(a) shows the detailed distribution of the predictions. Of the 9,711 normal records, 9,444 are classified correctly as true negatives and only 267 are mislabelled as attacks (false positives). This low false positive rate means that the system is unlikely to fatigue network administrators with unnecessary alarms. Of the 12,832 attack records, 7,812 are detected successfully (true positives) while 5,020 are missed (false negatives). The model therefore struggles with complex attack vectors that are under-represented in the training data, and its decision boundary is conservative.

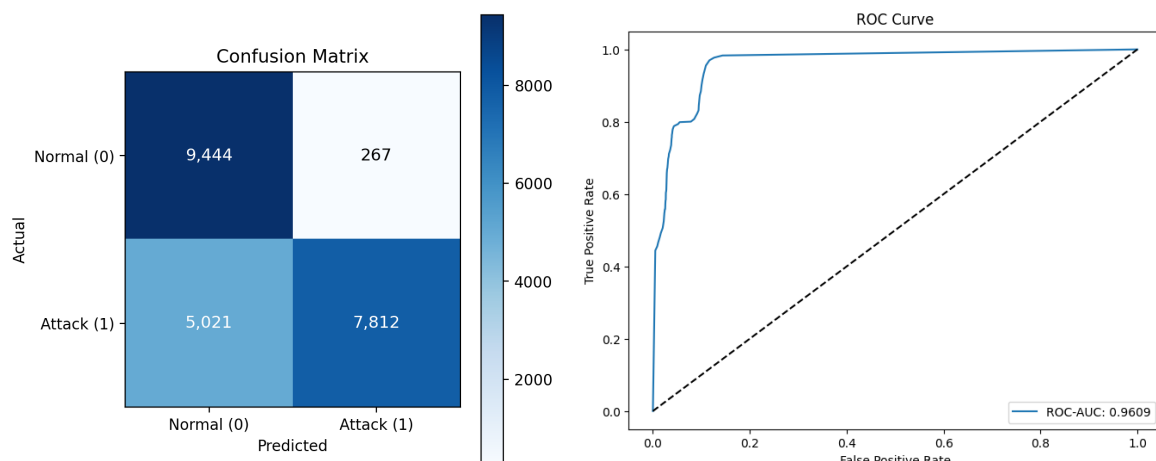


Figure 4. Random forest model on the KDDTest+ partition: (a) confusion matrix; (b) ROC curve (AUC = 0.9609).

The ROC curve constructed to assess the separating capacity of the model, and the area under it, are shown in Figure 4(b). The AUC of 0.9609 indicates that the model ranks attack and normal records apart from one another with statistically strong performance. The curve rises steeply towards the upper-left corner, showing that a high true positive rate is reachable at a low false positive rate. The contrast between this high AUC and the comparatively low recall in the classification report suggests that the probability scores produced by the model are correctly

ordered, and that classification performance could be improved further by moving the decision threshold away from its default value of 0.5.

In summary, the proposed random forest model offers high operational reliability through a minimised false positive rate, but raising its recall on complex attack vectors would require advanced feature engineering or the integration of a hybrid architecture.

4.2. Artificial neural network

Table 4 reports the classification performance of the proposed ANN on the KDDTest+ partition, where it reaches an accuracy of 79.85%. Examined per class, the model shows a high recall of 0.92 on normal traffic (class 0) while its precision for that class remains at 0.70. For attack traffic (class 1) the precision rises to 0.92 and the recall is 0.70. The model is therefore highly reliable when it flags an attack, but remains limited in the range of attack types it covers. The F1-score of 0.80 obtained for both classes indicates a more evenly balanced classification performance than that of the random forest.

Table 4. Classification report of the artificial neural network on the KDDTest+ partition.

Class	Precision	Recall	F1-score	Support
Normal (0)	0.70	0.92	0.80	9,711
Attack (1)	0.92	0.70	0.80	12,833
Accuracy	—	—	0.7985	22,544
Macro average	0.81	0.81	0.80	22,544
Weighted average	0.83	0.80	0.80	22,544
ROC-AUC	—	—	0.8797	—

The confusion matrix detailing the prediction distribution of the ANN is given in Figure 5(a). The model classifies 8,969 of the 9,711 normal records correctly as true negatives and mislabels 742 of them as attacks. Of the 12,833 attack records, 9,033 are detected successfully while 3,800 attack attempts are missed. Compared with the random forest, the ANN raises the number of detected attacks from 7,812 to 9,033, an improvement of 15.63%, but the number of false alarms rises in parallel from 267 to 742. The attack-capturing capacity of the ANN is therefore higher, while additional optimisation would be required to minimise its false positives.

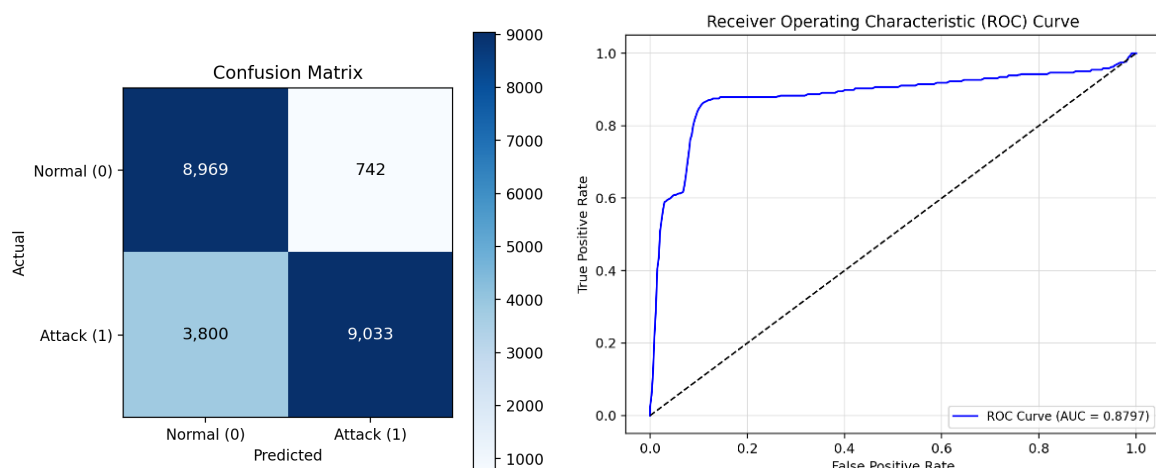


Figure 5. Artificial neural network on the KDDTest+ partition: (a) confusion matrix; (b) ROC curve (AUC = 0.8797).

The ROC curve of the ANN and its AUC value of 0.8797 are shown in Figure 5(b). Although the AUC is lower than that of the random forest, at the operating point selected here the network offers a more even trade-off between recall and precision. The behaviour of the curve shows that the model achieves a rapid increase in the true positive

rate at low false positive rates (below 0.1) and then saturates into a flatter regime. This indicates that the ANN captures certain attack types with very high success while its discriminative power falls off on the rarer and more complex types in the tail of the distribution. Even so, an AUC of 0.8797 confirms that the overall performance of the model is reliable.

In the light of these findings the ANN architecture has a high capacity for learning complex attack vectors, but it would benefit from support by hybrid architectures or advanced feature optimisation techniques in order to minimise false alarm rates.

4.3. Comparative analysis of training and test performance

When the performance of the two models is considered as a whole (Table 5), both architectures display a high learning capacity on the training partition, approaching 99%: the random forest reaches 99.99% and the ANN 99.82% training accuracy, which shows that both model known attack patterns very successfully. On data they have never encountered, however, a marked decline in generalisation is observed, with accuracy falling to 76.55% and 79.85% respectively.

Table 5. Comparison of the training and test performance of the two models.

Model	Training accuracy (KDDTrain+)	Test accuracy (KDDTest+)	Generalisation gap
Random forest	99.99%	76.55%	−23.44 pt
Artificial neural network	99.82%	79.85%	−19.97 pt

The root causes of this loss of performance, and the behavioural differences between the two models, can be read more clearly from the confusion matrices presented in Table 6. When the error types are analysed, the random forest is found to produce only 267 false alarms on the test partition, whereas that figure rises to 742 for the ANN. This conservative and stable behaviour of the random forest is a critical operational advantage for real-time systems, where false alarms interrupt the flow of legitimate traffic. On the threat-capturing side, however, both models run into a similar bottleneck: the random forest fails to detect 5,020 attack attempts and the ANN 3,800. That the number of missed attacks is this high in both models, and of the same order of magnitude, indicates that two factors act together. The first is overfitting to the training distribution. With unrestricted tree depth, a minimum leaf size of one and bootstrap resampling disabled, the random forest fits the training partition almost exactly, and the neural network reached a validation accuracy of 99.87% on a split drawn from the same balanced training distribution; a model that scores this highly on held-out data drawn from the training distribution, yet loses some twenty percentage points on KDDTest+, has fitted its decision boundary to that distribution rather than to the phenomenon. The second factor, to which the confusion matrices point more directly, is the presence in KDDTest+ of attack variants of a zero-day character that are absent from the training set; because the two models fail on comparable numbers of attacks despite their very different inductive biases, a substantial part of the gap cannot be attributed to either architecture alone.

Table 6. Comparison of the confusion matrices of the two models on the KDDTest+ partition.

Model	True negatives (normal traffic detected)	False positives (false alarms)	False negatives (missed attacks)	True positives (attacks detected)
Random forest	9,444	267	5,020	7,812
Artificial neural network	8,969	742	3,800	9,033

A word is needed on the apparent tension between the two headline figures: the neural network records the higher test accuracy (79.85% against 76.55%) while the random forest records the higher ROC-AUC (0.9609 against 0.8797). The two quantities answer different questions. Accuracy is measured at a single operating point — here the default decision threshold of 0.5, at which a predicted probability above 0.5 is labelled an attack — and therefore reports how well that particular threshold happens to be placed. ROC-AUC is computed across all thresholds and

reports how reliably a model ranks attack records above normal ones, independently of where the cut is made. The random forest separates the two classes more reliably in this ranking sense, but at a threshold of 0.5 its decision boundary sits in a conservative region, accepting a recall of 0.61 on the attack class in exchange for only 267 false alarms. The default threshold of the neural network happens to fall closer to the balance point of its own score distribution, which is why it converts a weaker ranking into a higher accuracy. The practical implication is that the accuracy reported for the random forest understates what the model can deliver: because the ordering of its scores is sound, calibrating the threshold on a validation partition would be expected to raise its recall substantially at a modest cost in false alarms, whereas the same procedure would yield less for the neural network. The threshold was left at 0.5 for both models so that the comparison would rest on the classifiers themselves rather than on a tuning step, and threshold calibration is identified in Section 6 as an immediate direction for further work.

The random forest therefore attains detection rates comparable to those of the ANN against unknown attacks while producing a markedly lower false alarm rate, which makes it the more sustainable and reliable architecture from the perspective of network security management. The ANN, conversely, is preferable in deployments where the cost of missing an attack outweighs the cost of raising an alarm.

4.4. Model serialisation and integration

Having completed training and testing successfully, both models were serialised so that they could be integrated into a real-time system. The formats given in Table 7 were selected so that the in-memory state of each model could be stored and reloaded in a production environment. The random forest was stored in the .pkl (joblib) format standard in the Python ecosystem, while the native Keras archive format (.keras) was used for the neural network so that its architecture and weights would be preserved without loss. Serialisation allows a trained model to be reloaded and queried without being retrained on each invocation.

Table 7. Storage formats of the trained models.

Model architecture	Storage format	Purpose
Random forest	.pkl (joblib)	scikit-learn model serialisation
Artificial neural network (MLP)	.keras (Keras v3 archive)	Keras/TensorFlow model architecture and weights

5. Design and Implementation of the Real-Time Anomaly Detection System

In order to demonstrate the practical applicability of the models developed in this study, the trained algorithms were integrated into an anomaly detection system built on the Flask micro web framework. The system architecture consists of two layers: a backend service that listens to and analyses network traffic, and a frontend interface through which the results are visualised. This hybrid structure is intended to give network administrators a dynamic, real-time security monitoring capability that goes beyond static log analysis.

5.1. Interface design and visualisation

The frontend of the system was designed with Tailwind CSS in a user-friendly and responsive layout. The interface is equipped with dynamic components so that operators can monitor network security from a single, integrated view. As Figures 6 and 7 show, the system reports its status to the user when it is started and signals that the analysis process has been triggered.

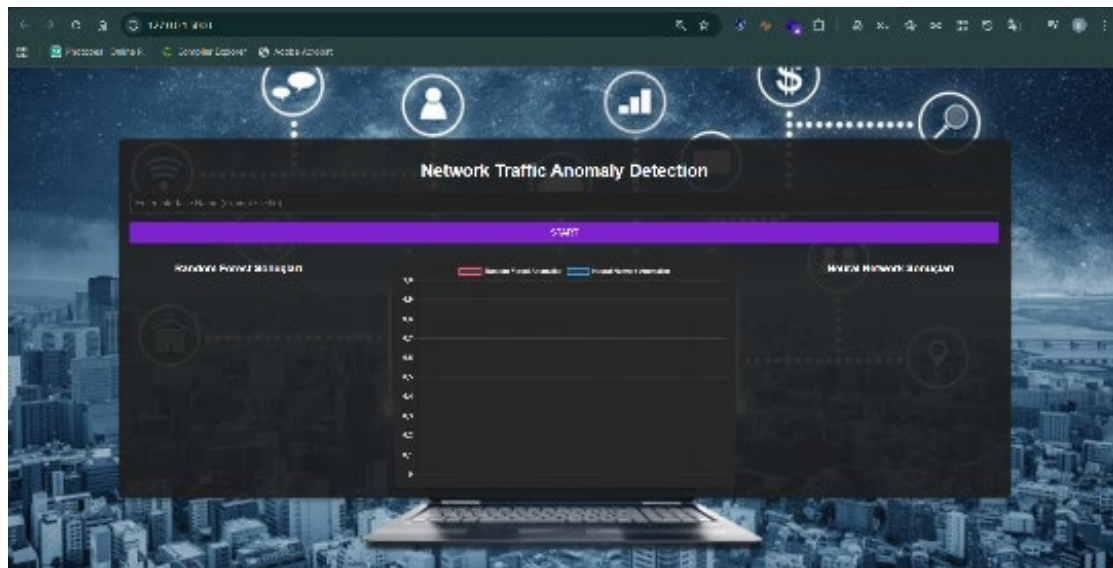


Figure 6. State of the web application interface before packet capture is started.

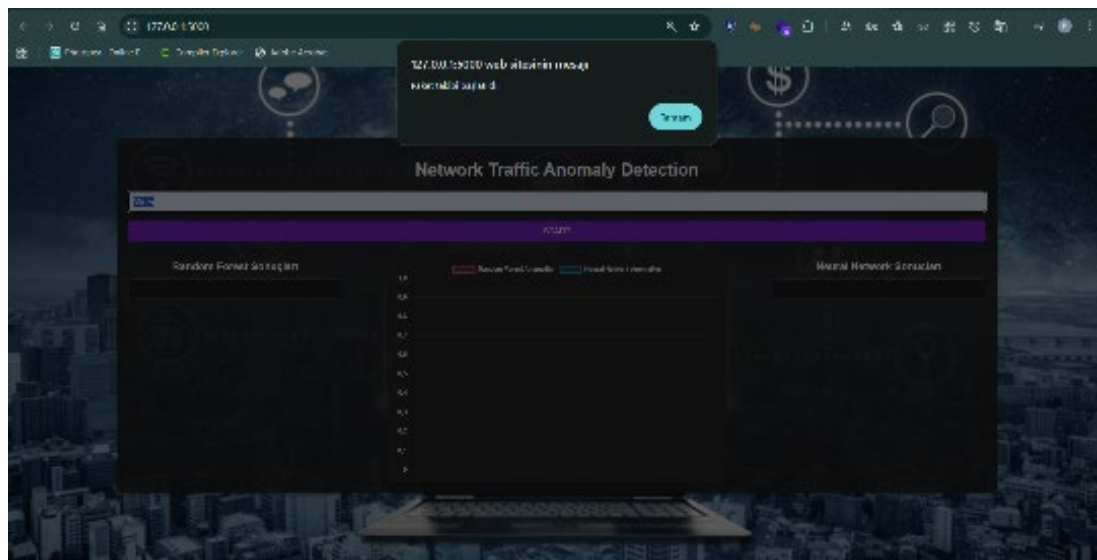


Figure 7. State of the interface with the notification indicating that packet capture has begun.

The visualisation module displays on screen, simultaneously, the classification produced by the random forest and by the ANN for every packet entering the system, labelled either “Normal” or “Anomaly”. So that the performance differences between the two models can be observed, the predictions are also presented comparatively through time-series charts and tables (Figure 8). This approach gives decision makers both an instantaneous picture of the threat situation and the means to test the stability of the models in a live environment.

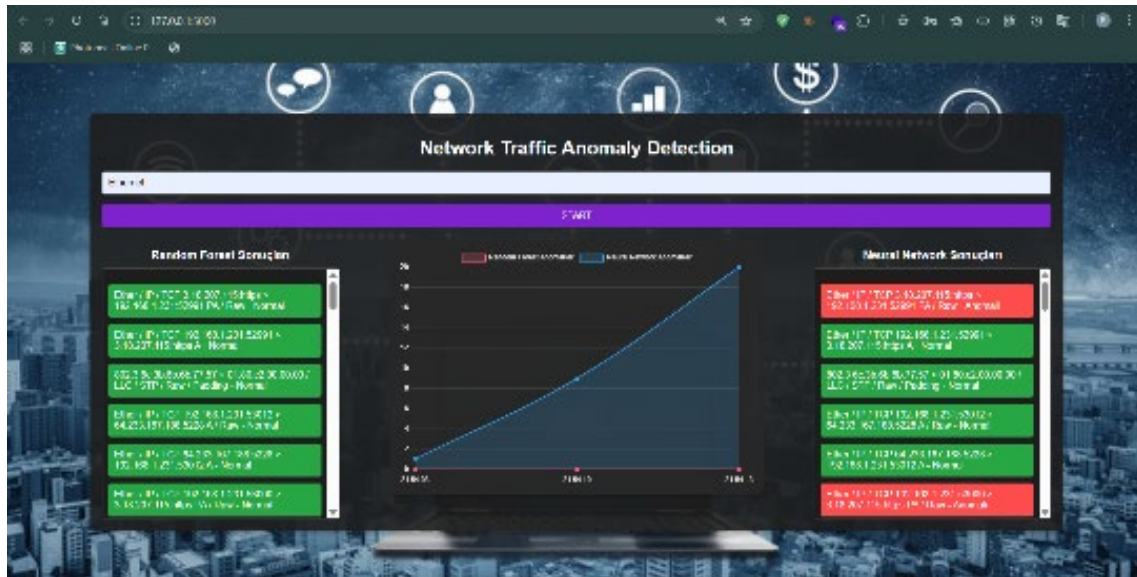


Figure 8. Real-time packet capture and the comparative display of model output in dynamic tables and charts.

5.2. Real-time traffic analysis and data flow architecture

The prototype is served by a separate pair of models, trained on the forty-one raw attributes without one-hot encoding and without the hyperparameter search described in Section 3.3, rather than by the models evaluated in Section 4; the two pairs are not interchangeable, because the models of Section 4 expect the 122-dimensional encoded vector. The operational strength of the system rests on a packet capture (sniffing) mechanism running in the background and built on the Scapy library. In real-time mode the system listens on a designated network interface, captures raw data packets and passes them into a real-time preprocessing pipeline. The captured packets are then assembled into a feature vector in the NSL-KDD attribute order and passed to the models held in memory. The models return a probability for each vector, and a threshold of 0.5 converts it to a Normal or Anomaly label. The predictions produced by the models are held in an asynchronous queue and served to the web interface through the Flask API. This data flow architecture allows potential anomalies in the network traffic to be detected on a millisecond timescale and written into the dynamic tables of the interface.

In the present prototype the mapping between a captured packet and the NSL-KDD attribute set is deliberately minimal, and its limits should be stated plainly. Of the forty-one attributes, only `src_bytes` is derived from the traffic, as the total length of the captured packet; the remaining attributes are supplied as fixed placeholder values. The connection-level counters — duration, count, `src_count` and the host-based statistics — are defined over completed connections and over a window of one hundred connections, and cannot be computed from a packet in isolation without a flow-tracking layer, which the prototype does not implement; they are set to zero. The content attributes obtained by inspecting the payload of a completed session, such as `hot`, `num_failed_logins` and `num_compromised`, are set to zero for the same reason. The categorical attributes `protocol_type`, `service` and `flag` are passed as constants rather than being read from the packet header and one-hot encoded.

The consequence is that the prototype demonstrates the capture, queueing, inference and visualisation path of a deployed detector, but does not reproduce the detection performance reported in Section 4: a feature vector in which a single attribute varies cannot carry the information the models were trained to exploit. Section 5 should therefore be read as a description of the system architecture and its interface, and the figures in it as a record of that interface in operation, rather than as an evaluation of live detection. Closing this gap requires a flow-assembly layer that aggregates packets into connections and then reapplies the one-hot encoder, the logarithmic transformation and the min-max scaler fitted on the training partition; this is identified in Section 6 as the first priority for further work.

6. Discussion and Conclusion

This study compared a random forest and a multilayer perceptron artificial neural network for the problem of anomaly detection in network traffic, using the NSL-KDD dataset under a shared preprocessing pipeline. The comprehensive preprocessing, feature analysis and normalisation steps contributed directly to the generalisation capability of both models. The experimental results show that both models achieve an accuracy approaching 99% on the training partition, but that the accuracies obtained on the test partition (76.55% for the random forest and 79.85% for the ANN) exhibit a generalisation gap, caused in part by overfitting to the training distribution and in part by the complex structure of the test data and by the presence of attack types that are unknown to the models.

The error analysis shows that the two architectures differ less in overall accuracy than in the kind of error they make. With 267 false positives against 742, the random forest offers a markedly lower false alarm rate and therefore a clear advantage in operational reliability, while the ANN detects 1,221 more attacks. Which model is preferable consequently depends on the relative cost of a false alarm and a missed attack in the deployment context. The SMOTE step removed the class imbalance of the training partition and increased the capacity of the models to learn rare attack types, and the random forest attained a ROC-AUC of 0.9609. Taken together, these findings indicate that the stability arising from the ensemble structure of the random forest makes it the more suitable architecture for real-time intrusion detection systems that require a low false alarm rate. The study further demonstrated that the models can be integrated into a real-time IDS prototype through a Flask-based web service, with traffic capture provided by the Scapy library and predictions visualised dynamically in the interface, subject to the mapping limitations set out in Section 5.2.

Three limitations qualify these results. First, although NSL-KDD is a corrected benchmark, it derives from traffic captured more than two decades ago and does not represent current threat profiles. Second, the prototype does not reconstruct the connection-level NSL-KDD feature space from live packets: only *src_bytes* is derived from the traffic and the remaining attributes are held at placeholder values, so the figures reported here characterise offline performance on KDDTest+ and say nothing about live detection accuracy. Third, the decision threshold was left at its default value of 0.5 in both models; the gap between the random forest's ROC-AUC of 0.9609 and its recall of 0.61 indicates that threshold calibration alone could shift the operating point substantially.

Against these limitations, three directions are identified for future research.

Use of modern datasets. In order to reflect present-day network threat profiles more accurately and to improve the generalisation capability of the models, the use of more recent, larger corpora containing current attack vectors, such as CICIDS2017, is intended in place of NSL-KDD.

Exploration of deep learning architectures. In order to capture the temporal dependencies between traffic packets more effectively, the performance benefits of recurrent architectures such as long short-term memory (LSTM) and gated recurrent unit (GRU) networks will be investigated in place of the feed-forward model used here.

Optimisation of resource efficiency for edge computing. So that the developed system can be deployed in low-resource environments or on network edge devices, work will be carried out on lightening the models through quantisation and on optimising the system with respect to energy consumption.

Flow-level feature assembly. The most immediate requirement is a flow-tracking layer that aggregates captured packets into connections, computes the time-based and host-based counters over a sliding window, and reapplies the encoder and scalars fitted on the training partition, so that the deployed models receive the representation on which they were trained.

CRedit authorship contribution statement. C.B.HAKAY: Conceptualization, Methodology, Software, Investigation, Writing – original draft. S.GENÇ: Supervision, Formal analysis, Validation, Supervision, Writing – review & editing.

Funding. This work was supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK) under the 2209-A programme, project no. 1919B012409245.

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 and reproducibility. The NSL-KDD dataset analysed in this study is publicly available from the Canadian Institute for Cybersecurity [7]. The models were built with scikit-learn, TensorFlow/Keras and imbalanced-learn under Python 3. The random seed was fixed to 42 for the over-sampling and for the random forest and its hyperparameter search; the

initialisation of the neural network was not seeded, so its weights are not reproducible bit for bit, although the reported metrics were stable across runs.

Declaration of generative AI use. In the preparation of this article, we utilised generative artificial intelligence for support—including language editing, translation, coding assistance and literature reviews. Generative AI tools are not listed as authors; the authors take full responsibility for every claim, analysis and citation in the paper.

References

- [1] Y. Xin et al., “Machine learning and deep learning methods for cybersecurity,” *IEEE Access*, vol. 6, pp. 35365–35381, 2018, doi: 10.1109/ACCESS.2018.2836950.
- [2] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, “A survey of deep learning-based network anomaly detection,” *Cluster Comput.*, vol. 22, no. 1, pp. 949–961, 2019, doi: 10.1007/s10586-017-1117-8.
- [3] I. Ahmad, M. Basher, M. J. Iqbal, and A. Rahim, “Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection,” *IEEE Access*, vol. 6, pp. 33789–33795, 2018, doi: 10.1109/ACCESS.2018.2841987.
- [4] A. Moldagulova and R. B. Sulaiman, “Using KNN algorithm for classification of textual documents,” in *Proc. 8th Int. Conf. Inf. Technol. (ICIT)*, Amman, Jordan, 2017, pp. 665–671, doi: 10.1109/ICITECH.2017.8079924.
- [5] University of California, Irvine, “KDD Cup 1999 task description.” [Online]. Available: <https://kdd.ics.uci.edu/databases/kddcup99/task.html> (accessed Aug. 22, 2026).
- [6] M. Tavallae, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl. (CISDA)*, Ottawa, ON, Canada, 2009, pp. 1–6, doi: 10.1109/CISDA.2009.5356528.
- [7] Canadian Institute for Cybersecurity, “NSL-KDD dataset,” University of New Brunswick. [Online]. Available: <https://www.unb.ca/cic/datasets/nsl.html> (accessed Aug. 8, 2026).
- [8] L. Dhanabal and S. P. Shantharajah, “A study on NSL-KDD dataset for intrusion detection system based on classification algorithms,” *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, no. 6, pp. 446–452, 2015.
- [9] M. C. Belavagi and B. Muniyal, “Performance evaluation of supervised machine learning algorithms for intrusion detection,” *Procedia Comput. Sci.*, vol. 89, pp. 117–123, 2016, doi: 10.1016/j.procs.2016.06.016.
- [10] B. Ingre and A. Yadav, “Performance analysis of NSL-KDD dataset using ANN,” in *Proc. Int. Conf. Signal Process. Commun. Eng. Syst. (SPACES)*, Guntur, India, 2015, pp. 92–96, doi: 10.1109/SPACES.2015.7058223.
- [11] S. Revathi and A. Malathi, “A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection,” *Int. J. Eng. Res. Technol.*, vol. 2, no. 12, pp. 1848–1853, 2013.
- [12] MIT Lincoln Laboratory, “1998 DARPA intrusion detection evaluation dataset.” [Online]. Available: <https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset> (accessed Aug. 22, 2026).
- [13] L. Yu, R. Zhou, R. Chen, and K. K. Lai, “Missing data preprocessing in credit classification: One-hot encoding or imputation?,” *Emerg. Markets Finance Trade*, vol. 58, no. 2, pp. 472–482, 2022, doi: 10.1080/1540496X.2020.1825935.
- [14] A. Singh, J. Amutha, J. Nagar, S. Sharma, and C.-C. Lee, “LT-FS-ID: Log-transformed feature learning and feature-scaling-based machine learning algorithms to predict the k-barriers for intrusion detection using wireless sensor network,” *Sensors*, vol. 22, no. 3, art. no. 1070, 2022, doi: 10.3390/s22031070.
- [15] M. Shantal, Z. Othman, and A. A. Bakar, “A novel approach for data feature weighting using correlation coefficients and min-max normalization,” *Symmetry*, vol. 15, no. 12, art. no. 2185, 2023, doi: 10.3390/sym15122185.
- [16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002, doi: 10.1613/jair.953.
- [17] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
- [18] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, 2015. [Online]. Available: <https://arxiv.org/abs/1412.6980>