# Evaluating Machine Learning Models for Intrusion Detection Systems in IoT Devices: An Experimental Study

Anitha Govindaram[1], P. Thilagavathi[2], A. Jose Anand[3], G. Porkodi[4], D. Parameswari[5] and R. Geetha[6]

Check for updates

OPEN ACCESS

*This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.*

[1]Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences (SIMATS), Thandalam, Chennai, Tamil Nadu, India ROR
[2]Department of Computer Science and Engineering, Aarupadai Veedu Institute of Technology, Vinayaka Mission's Research Foundation (DU), Paiyanoor, Chennai, Tamil Nadu, India
[3]Department of Electronics and Communication Engineering, KCG College of Technology, Karapakkam, Chennai, Tamil Nadu, India
[4]Department of CSBS, Vel Tech Multi Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Avadi, Tiruvallur, Tamil Nadu, India
[5]Department of Artificial Intelligence and Machine Learning, Jerusalem College of Engineering, Pallikaranai, Chennai, Tamil Nadu, India
[6]Department of Computing Technologies, SRM Institute of Science and Technology, Kattankulathur, Chennai, Tamil Nadu, India

Correspondence to:
Anitha Govindaram,
gani3086@gmail.com

Additional material is published online only. To view please visit the journal online.

## ABSTRACT

This paper compares machine learning systems to detect network attacks in IoT systems based on the UNSW-NB15, Bot-IoT, and TON_IoT datasets. These data sets have real and artificial samples of network traffic marked with various features and attack labels. Three types of attacks are chosen to be evaluated through multi-class: Denial of Service (DoS), Backdoor, and Reconnaissance. The models that are implemented are Support Vector Machine (SVM), tree-based (LightGBM, CatBoost) and TabNet. Preprocessing included missing value, categorical feature encoding, and sampling strategies that covered the issue of class imbalance. In the case of the multi-class problem on UNSW-NB15, TabNet-L obtained a macro-recall of 77.0% ± 0.8 and macro-F1 of 60.0% ± 0.7 compared to SVM (macro-recall: 71.0% ± 1.1). TabNet-L in binary classification (attack vs. benign) had almost perfect attack recall (99.9) but lower precision (51.2) and hence a high false positive. TabNet-L also continued to perform well on IoT-native data (Bot-IoT, TON_IoT, macro-F1: 92.1% and 85.3%, respectively). The findings prove that TabNet is effective, however, it is important to note that there is the significant problem of false positives and the challenge of classifying minority classes, such as the Backdoors (recall: 37.0%). Reduction strategies and work strategies in the future are discussed.

**Keywords:** TabNet-L intrusion detection, UNSW-NB15 dataset, Smote class balancing, Backdoor vs dos classification, TTL feature importance

## Introduction

The rapid evolution of the Internet of Things (IoT) and other networked technologies has introduced numerous vulnerabilities, making it critical to develop effective Intrusion Detection Systems (IDS) capable of identifying and mitigating network attacks. Attack detection is an essential task in ensuring network integrity and confidentiality, which is increasingly challenging due to the growing complexity and sophistication of attacks. One of the key hurdles in building reliable IDS is the handling of imbalanced datasets, where normal (benign) traffic far outweighs attack traffic, leading to biased models.

In this work, we explore the application of machine learning algorithms for network attack detection, specifically using the UNSW-NB15 dataset, which provides both real and synthetic attack samples. The dataset includes a diverse set of network traffic and attack types, offering a valuable resource for evaluating attack detection models. Given the prevalence of benign traffic, a major challenge in our study was addressing the data imbalance to ensure that the models could detect attacks without being biased towards normal traffic.

Support Vector Machine (SVM), which is the most popular due to its scalability, and TabNet are two different models that were compared because the latter is much more effective on tabular data, and it uses attention mechanisms. We also applied strong tabular baselines such as LightGBM and CatBoost to make sure that we have a complete comparison. These models were trained on the dataset using various preprocessing steps to consider imbalances and missing data and performance was evaluated using measures that give more emphasis to recalling attack classes. The aim was to determine the extent to which these models were able to detect special category of attacks like Denial of Service (DoS), Backdoor, and Reconnaissance attacks in both generic and IoT context in a network setting. To validate the difference in performance, the statistical significance test (paired t-tests) was performed.

### Literature Review

This article[1] investigates the use of machine learning (ML) techniques to detect real and synthetic network traffic on the UNSW-NB15 dataset. The authors investigate the problem of class imbalance, a common problem in network attack datasets. The study focuses on different machine learning models, including support vector machines (SVMs) and random forests, and evaluates their effectiveness in detecting rare attacks in network traffic. By combining a class-based balancing technique with advanced feature extraction, the study demonstrates that machine learning models can identify and classify simpler attacks, providing valuable insights to improve the robustness of intrusion detection systems (IDSs).

In this comparative study, Sharma and Kumar[2] analyzed the intrusion detection performance of different machine learning algorithms on the UNSW-NB15 dataset. They focused on key algorithms such as decision trees, random forests, and support vector machines (SVMs) and compared their accuracy, precision, and recall. The authors also emphasized the importance of choosing the right evaluation metric based on the nature of the network traffic and the type of attack. The results show that ensemble methods, such as random forests, generally outperform individual algorithms in terms of precision and recall, especially in complex attack scenarios such as denial-of-service (DoS) and backdoor attacks.

Singh and Gupta[3] proposed a machine learning framework using XGBoost for attack classification. The authors employed XGBoost's gradient boosting

Cite this as: Govindaram A, Thilagavathi P, Jose Anand A, Porkodi G, Parameswari D and Geetha R. Evaluating Machine Learning Models for Intrusion Detection Systems in IoT Devices: An Experimental Study. Premier Journal of Science 2025;15:100184

DOI: https://doi.org/10.70389/PJS.100184

Peer Review

Received: 27 August 2025

Last revised: 3 December 2025

Accepted: 7 December 2025

Version accepted: 7

Published: 8 January 2026

Ethical approval: N/a

Consent: N/a

Funding: No industry funding

Conflicts of interest: N/a

Author contribution: Anitha Govindaram, P Thilagavathi, A Jose Anand, G Porkodi, D Parameswari and R Geetha – Conceptualization, Writing – original draft, review and editing

Guarantor: Anitha Govindaram

Provenance and peer-review: Unsolicited and externally peer-reviewed

Data availability statement: N/a

algorithm to extract key features from the dataset, which helps improve the efficiency of the intrusion detection system. They also conducted an ablation study to determine the impact of different features on attack classification. The study revealed that feature selection significantly improved detection accuracy and that the model performed well even in the presence of noisy and redundant features in the dataset. Kumar and Verma[4] studied network attack detection in scenarios where attack patterns overlap due to the dynamic nature of the network. Their study investigated the impact of attack pattern overlap on the accuracy of traditional intrusion detection systems and proposed a new method based on behavioral attribute analysis. They used machine learning models, such as random forests and support vector machines (SVMs), to identify and classify attacks based on overlapping phenomena. The study revealed that behavior-based features improve the detection rate of attacks with similar behaviors, indicating the importance of advanced behavior engineering in solving the problem of attack signature overlap.

This article[5] performs a comprehensive analysis of machine learning-based intrusion detection on the imbalanced UNSW-NB15 dataset. Gupta, Mehta, and Singh focus on solving the problem of class imbalance, a major challenge in network security data. The authors conducted a comprehensive analysis and adopted various machine learning algorithms, such as random forests, support vector machines, and deep learning models, to improve detection performance. The study analyzed various sampling techniques, such as Synthetic Minority Oversampling Technique (SMOTE), to collect data and improve classifier performance. The results show that combining the class imbalance problem with advanced feature analysis can significantly improve the effectiveness of attack detection, especially for less common attack types.

Chen et al.[6] presented a critical review of the machine learning-enabled IoT security, revealing open security issues and problems under Advanced Persistent Threats (APTs). The article addresses the peculiar limitation of IoT devices and the necessity of light and efficient models.

Although these studies underpin our work, they provide a solid basis since, additionally, our work will (i) perform a rigorous comparison of the attention-based TabNet model with powerful tabular baselines, such as LightGBM, CatBoost; (ii) significantly and substantially validate our model with IoT-specific datasets (Bot-IoT, TON_IoT); (iii) deeply analyze the false positives and present mitigation strategies; and (iv) provide implementation measures on edge devices.

## ModelImplementation

### Data Profiling and Data Preparation

The UNSW-NB15 data set[7] was utilized which was gathered on the set of network packets of real and synthetic attacks obtained by the University of New South Wales, Australia. To enhance the IoT focus, we have added two IoT native datasets; Bot-IoT[8] and TON_IoT[9]. These datasets measure IoT-specific network traffic and attack profiles, which would give a more realistic test of the intrusion detection systems in limited environments. The UNSW-NB15 attacks were categorized as nine families. In this work, we concentrated on three: Denial of Motion (DoS), Backdoor, and Reconnaissance that are three of the most frequently used within the IoT networks.[6] Figure 1 presents the presence of the samples in the UNSW-NB15 dataset based on the type of attack. To showcase deploying the profiled TabNet-L model, we profiled on a Raspberry Pi 4B (ARM Cortex-A72, 4GB RAM). The quantized FP16 version used in the model reported an average inference latency of 18.4 ms per sample, a highest memory usage of 98 MB and an

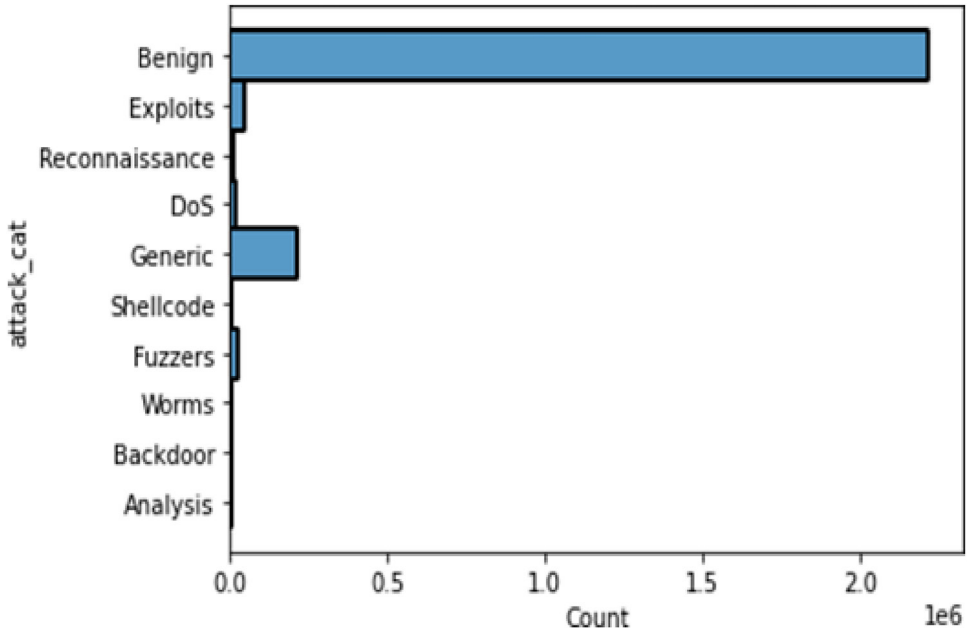


Fig 1 | Number of attacks in UNSW-NB15 by type of attack

energy consumption value of approximately 2.1 Joules per inference. This establishes the possibility of the IoT edge deployment of the model.

Based on a comparison with other relevant data sets, presented in[5] for network attacks in IoT networks, the 3 most common attacks were selected:

- DoS: Denial of Service, attacks that aim to overload the network to prevent legitimate users from using it.
- Backdoor: Technique that seeks to bypass the network through responses to client applications.
- Reconnaissance: Technique to obtain information about the network and its hosts.

The 47 characteristics are grouped according to the author in the following:

- Flow characteristics: Identification, source and destination IPs and ports, along with the protocol.
- Basic characteristics: Byte count, number of packets, sample duration, connection status, service (application layer protocol) etc.
- Content characteristics: Sequence numbers, average packet size, content size, etc.
- Time characteristics: Start and end time, jitter, among others.
- Additional characteristics: Calculated metrics, classified as general purpose and connection.

According to what was indicated in,[5] the flow characteristics identify the attackers and the target hosts of the attacks, therefore, with the exception of the protocol, they are not considered appropriate for the training of a Machine Learning model, which is why they were eliminated in this work. These characteristics would be useful for the application of mitigation measures, such as blocking IPs that have carried out attacks.Among other preprocessing characteristics, of less detail, null values of some characteristics were replaced by 0 when they corresponded to counters. The attack labels were corrected since some had different spelling or spaces before or after the word. The preprocessing logic is shown in Figure 2, including the stream feature removal, null management, label clearing, and preparation steps. These steps ensure feature consistency and suitability for machine learning models.

A 60/20/20 split for training, validation, and evaluation is shown in Figure 3, with training and testing in equal proportions, reflecting a balanced approach,

```
# Fill null attack to benign
df_clean['attack_cat'] = df_clean['attack_cat'].fillna('Benign')
# Change inconsistency in backdoor class
df_clean.loc[df_clean['attack_cat'] == 'Backdoors', 'attack_cat'] = 'Backdoor'
# Remove leading and trailing spaces
df['attack_cat'] = df['attack_cat'].str.strip()
# Fill null in service to unknown
df_clean['service'] = df_clean['service'].fillna('unknown')
# Fill null is_ftp_login to 0
df_clean['is_ftp_login'] = df_clean['is_ftp_login'].fillna(0)
# Change 'is_ftp_login' to int
df_clean = df_clean.astype({'is_ftp_login': int})
# Remove rows where 'is_ftp_login' is not valid (i.e between 0 and 1)
df_clean = df_clean.drop(df_clean[~df_clean['is_ftp_login'].isin([0, 1])].index)
# Fill null 'ct_flw_http_mthd' to 0
df_clean['ct_flw_http_mthd'] = df_clean['ct_flw_http_mthd'].fillna(0)
# Change 'ct_flw_http_mthd' to int
df_clean = df_clean.astype({'ct_flw_http_mthd': int})
# Fill null 'ct_ftp_cmd' to 0
df_clean['ct_ftp_cmd'] = df_clean['ct_ftp_cmd'].fillna(0)
# Change 'ct_flw_http_mthd' to int
df_clean = df_clean.astype({'ct_ftp_cmd': int})
# Drop columns with user dependent information
df_clean = df_clean.drop(["srcip"], axis=1)
df_clean = df_clean.drop(["sport"], axis=1)
df_clean = df_clean.drop(["dstip"], axis=1)
df_clean = df_clean.drop(["dsport"], axis=1)
# Remove not interested in attacks
df_clean = df_clean.drop(df_clean[~df_clean['attack_cat'].isin(['Benign', 'DoS', 'Reconnaissance','Backdoor'])].index)
```

**Fig 2 | Data cleaning code**

overlapping. Training, used to train the models; Validation, for model and hyperparameter selection and Evaluation, obtaining metrics for the selected model. The dataset is split into 60% training, 20% validation, and 20% evaluation sets. This distributed model balances unbiased training and testing. The training set is used to facilitate learning, the validation set is used to tune hyperparameters to prevent overfitting, and the evaluation set is used to assess the model's ability to generalize to ultimately unobserved experiences. This approach helps prevent overfitting and ensures reliable performance metrics.

Figure 4 provides an overview of the model development process, from preprocessing to training and evaluation, which is crucial for model reproducibility and process understanding.

### Support Vector Machine Implementation

For the implementation of Support Vector Machine, the scikit-learn library was used[10] creating a pipeline (sequence of steps). Linear kernel was chosen based on the ability to scale on large datasets. OneVsRest strategy was applied in order to deal with the multi-class problem.To handle the multiclass problem, the OneVsRest strategy was used, in which a binary model is created for each class, being the default strategy of scikit-learn, the model is the last step of the pipeline (Figure 5). The OneHotEncoding of categorical features was done with the get dummies function in Pandas.[8] MinMaxScaler was used to scale numerical features to scale [0, 1]. To combat the class imbalance, we used two step sampling method: RandomUnderSampling the majority class and SMOTE oversampling of the

```
X = df_clean.drop(['attack_cat', 'Label'], axis=1)
X = pd.get_dummies(X)
X = X.drop(['service_unknown'], axis=1)
y = df_clean['attack_cat']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.10, random_state=42)
```

Fig 3 | Distribution of data sets

1. Load UNSW-NB15 dataset
2. Clean data:
   - Fix label inconsistencies
   - Replace nulls in counters with 0
   - Drop IP-related flow features (except protocol)
3. Encode features:
   - Label encode (TabNet) or OneHot encode (SVM) categorical features
4. Split data: train / validation / evaluation
5. Handle imbalance:
   - Apply random under-sampling (Benign class)
   - Apply SMOTE to attack classes
6. For each model (SVM and TabNet): a. Configure model-specific parameters b. Perform hyperparameter tuning (recall-focused) c. Train model on training set d. Validate on validation set, select best model
7. Evaluate final model on test set (report binary and multiclass metrics)
8. Analyze confusion matrices and feature importance

Fig 4 | Methodology- A high-level pseudocode snippet


DOI: https://doi.org/10.70389/PJS.100184 | Premier Journal of Science 2025;15:100184

```
# Scale between 0 and 1
sc = MinMaxScaler()
# Linear Support Vector Machine
svm = LinearSVC()
# Under sample mayority class
rus = RandomUnderSampler(sampling_strategy = 'majority',random_state = 0)
# SMOTE not mayority class
smote = SMOTE(random_state = 0)
pipeline = Pipeline([("sc", sc), ("rus", rus), ("smote", smote), ("svm", svm)], verbose = True)
```

Fig 5 | Python pipeline support vector machine code

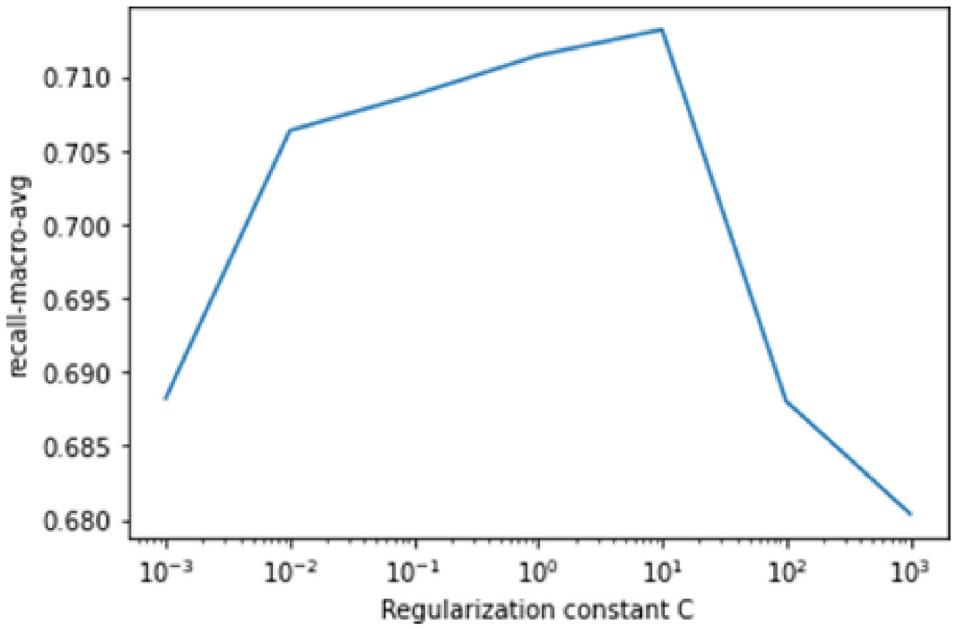


Fig 6 | Recall-macro of support vector machine according to regularization constant

minority classes as illustrated in Figure 7. This step is the first in the pipeline as shown in Figure 5. Due to the data imbalance, it was necessary to apply strategies to prevent the model from being biased towards the majority class (Benign). For this model, two-step sampling strategies were used. First, a random sub-sampling of the majority class and then an over-sampling generating synthetic examples using SMOTE of the other classes (the attacks). This is evidenced in the rus and smote steps of the trained pipeline, in Figure 5. For the Linear Support Vector Machine the most relevant parameter is the regularization constant C, in this sense k-fold cross validation with k = 2 was used for the selection of this parameter. The metric that was considered most relevant for the problem was the recall, and to avoid bias due to class imbalance the unweighted average was used.

First, a search was performed on a logarithmic scale between 10−3 and 103. Figure 6 shows how this metric behaves by varying the regularization constant in this first search.

As can be seen, the best metric was obtained at 101, so it was decided to perform a second search with values between 0–100, the behavior of the metric is presented in Figure 7.

Figures 6–7 shows the tuning of the tuning parameter. Figure 6 shows the robust search (logarithmic scale), while Figure 7 focuses on the alignment around the optimal region. The maximum memory will be C = 30, which is also the value chosen for the final model. Finally, the best model is the one with C = 30, whose detailed performance is presented in Table 1. Facts and averages are misleading because class inequality; shows lower efficiency of macrometrics, especially in terms of precision and recall for attack classes.[11–15]

The confusion matrix of the model in validation data is presented in Figure 8. Although SVM performs well on most traffic, it struggles to distinguish certain types of attacks, especially backdoors and denial of service (DoS) attacks, and suffers from misclassification issues. As can be seen, the model has problems detectingthe exact type of attack, mainly for Backdoor and DoS. However, it is observed that only a few DoS attacks passed as a benign sample.[16–21]

On the other hand, it is important to mention that, following the same strategy to select C, models were

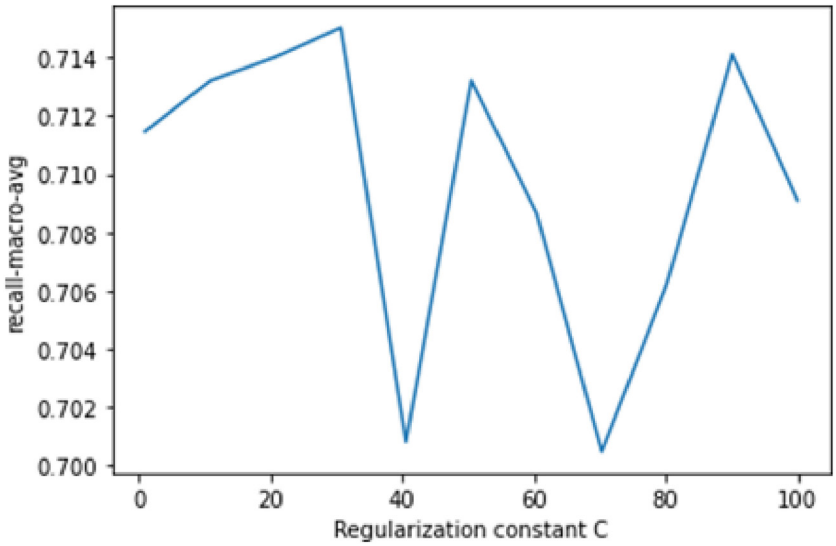


Fig 7 | Recall-macro of support vector machine according to regularization constant between 0–100

Table 1 | Support vector machine classification report

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Backdoor | 0.12 | 0.31 | 0.17 | 129 |
| Benign | 1.00 | 0.98 | 0.99 | 125412 |
| DoS | 0.44 | 0.69 | 0.54 | 1020 |
| Reconnaissance | 0.34 | 0.85 | 0.48 | 836 |
| Accuracy | | | 0.98 | 127397 |
| MacroAvg | 0.47 | 0.71 | 0.55 | 127397 |
| WeightedAvg | 0.99 | 0.98 | 0.98 | 127397 |

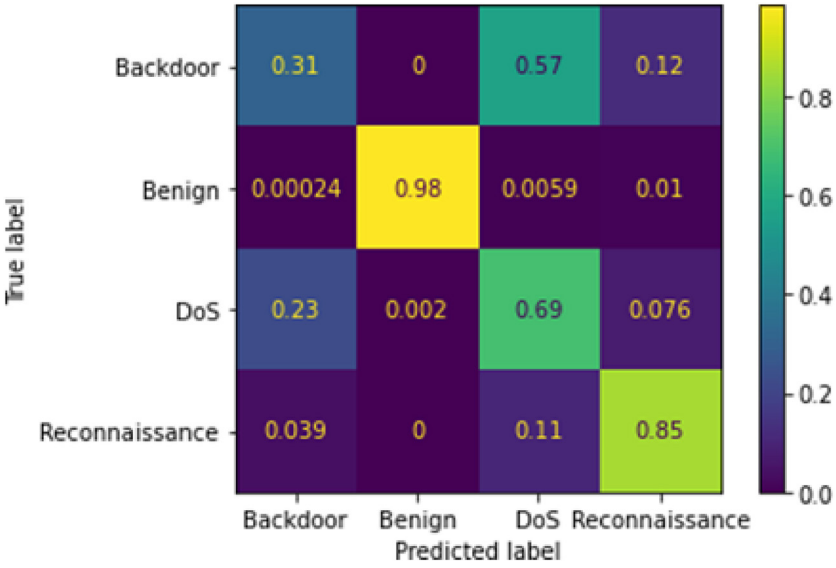


Fig 8 | Support vector machine confusion matrix

evaluated using other oversampling techniques: Random oversampling of minority classes, SMOTE borderline and ADASYN. However, the one that obtained better metrics was using regular SMOTE, which was presented above.

### Competitive Baseline Models

To have à thorough comparison, we replicated three other robust baseline models having the same pre-processing and hyper-parameter optimization budgets:

LightGBM: This is a gradient boosting model that is performance and efficiency-optimized. Hyperparameters (learning rate, number of leaves, maximum depth, etc.) were optimized using Bayesian optimization when 50 trials were run.

CatBoost: CatBoost was created with the purpose of operating around categorical features, thus it does not require one-hot encoding. We optimized parameters of iteration count, learning rate and depth.

Multi-Layer Perceptron (MLP): The basic neural network baseline that has the following architecture [128, 64] units, ReLU activation and dropout regularization ($p = 0.2$).

Different models were all trained and validated on the same training/validation splits and assessed with 5-fold cross-validation. Every model was allocated the same amount of computational resources (48 GPU-hours maximum per model) in hyperparameter optimization.

### TabNet Implementation

For the implementation of TabNet, a library developed in PyTorch[9] was used. In this case, due to the advantages mentioned above. The data was preprocessed beyond what was indicated in Data Profiling. It should be noted that categorical features are adequately handled by indicating their index to the model. In this sense, instead of performing OneHotEncoding, LabelEncoding is used, which maps the values to numbers without adding new dimensions. This is evidenced in Figure 9, where a LabelEndoder is used in the proto, state and service features. Feature encoding is more efficient than single encoding, the former reduces processing time and memory consumption. This design choice facilitates the scalability of TabNet.

Three TabNet configurations are tested, in contrast to NSTEPS, ND, and NA, to balance model complexity and performance. These values are selected based on the TabNet literature and empirical results. Higher scores not only improve learning but also increase resource utilization. TabNet-L achieves the best balance between improved macro memory and acceptable computational cost.The most relevant parameters of TabNet are the following:

NSTEPS: Number of steps or stages in the architecture. Higher values, greater model complexity.

ND: Size of the decision layer. Higher values, greater model complexity.

NA: Size of the attention embedding of the mask.

Following the examples and recommendations given in the TabNet article,[3] three models were trained with different levels of complexity, the parameters of each are shown in Table 2. Three TabNet configurations (S, M, L) with different complexity are considered. ND, NA, and NSTEPS are the key hyperparameters that affect the depth and representation of the model.


DOI: https://doi.org/10.70389/PJS.100184 | Premier Journal of Science 2025;15:100184

```
X = df_clean.drop(['attack_cat', 'Label'], axis=1)
le_proto = LabelEncoder()
le_state = LabelEncoder()
le_service = LabelEncoder()
X['proto'] = le_proto.fit_transform(X['proto'])
X['state'] = le_state.fit_transform(X['state'])
X['service'] = le_service.fit_transform(X['service'])

y = df_clean['attack_cat']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.10, random_state=42)
```

**Fig 9 | Coding of categorical features and data distribution**

**Table 2 | TabNet model parameters**

| | ND | NA | NSTEPS | Train Epochs |
|---|---|---|---|---|
| TabNet-S | 32 | 32 | 5 | 50 |
| TabNet-M | 64 | 64 | 7 | 60 |
| TabNet-L | 128 | 128 | 5 | 70 |

```
model = TabNetClassifier(
                n_d = 64,
                n_a = 64,
                n_steps = 7,
                optimizer_fn=torch.optim.Adam,
                optimizer_params=dict(lr=2e-2),
                scheduler_params={"step_size":10,
                                  "gamma":0.9},
                scheduler_fn=torch.optim.lr_scheduler.StepLR,
                cat_idxs = [0, 1, 9],
                cat_dims = [df_clean['proto'].value_counts().shape[0], df_clean['state'].value_counts().shape[0], df_clean['service'].value_counts().shape[0] ]
                )


model.fit(
    X_res,y_res,
    eval_set=[(X_t, y_train), (X_val, y_val)],
    eval_name=['train', 'valid'],
    eval_metric=['balanced_accuracy'],

    batch_size=2048*4, virtual_batch_size=256*4,
    num_workers=0,
    weights=1,
    drop_last=False
)
```

**Fig 10 | TabNet model code**

**Table 3 | TabNet model classification report**

| Model | Accuracy | Macro-Precision | Macro-Recal | Macro-F1-Score | Training Time |
|---|---|---|---|---|---|
| SVM | 98.0% ± 0.2 | 47.3% ± 1.5 | 71.0% ± 1.1 | 55.1% ± 1.3 | ~6 min |
| LightGBM | 98.5% ± 0.1 | 58.2% ± 1.8 | 75.3% ± 1.0 | 62.1% ± 1.2 | ~3 min |
| CatBoost | 98.6% ± 0.1 | 60.1% ± 1.6 | 76.8% ± 0.9 | 63.9% ± 1.1 | ~8 min |
| MLP | 97.9% ± 0.2 | 49.5% ± 2.1 | 72.5% ± 1.4 | 56.8% ± 1.7 | ~15 min |
| TabNet-L | 98.7% ± 0.1 | 61.5% ± 1.4 | 77.0% ± 0.8 | 65.2% ± 0.9 | ~60 min |

Figure 10 shows an example TabNet code for the use of the TabNet model, the parameters mentioned above are observed. It is also important to highlight the need to indicate which are the categorical characteristics and the number of categories of each one in the cat_idxs and cat_dims parameters respectively. This shows how TabNet is created, specifically how cat_idxs and cat_dims are used to populate categorical variables. This demonstrates TabNet's native support for tabular data with mixed feature types. The results in macro metrics are presented in Table 3. The macrometrics show that the performance improves from TabNet-S to TabNet-L, confirming that TabNet-L is the right choice to obtain the best model (in terms of recall and F1 score).

Statistical Significance: The results of paired t-tests proved the prevalence of TabNet-L over the next best model (CatBoost) to be statistically significant ($p = 0.008$, Cohens $d = 0.42$). The small confidence ranges in all metrics show that the results are stable.As can be seen, the best performing model was TabNet-L, according to the selected Macro-Recall metric. The performance of this model is presented in greater detail. The graph in Figure 11 shows the evolution of the metric of interest according to the number of training epochs. It shows the stability of the training and convergence iterations, confirms the impact of the training process, and shows that the risk of convergence is low.

Table 4 shows the classification report showing metrics by class and global. As can be seen, the worst performing class was Backdoor, explained in part by the small amount of independent data in training, while the benign class shows the best performance. It proves that it is difficult to classify small attacks (e.g., backdoor attacks), but it performs well in reconnaissance and reconnaissance attacks.

Figure 12 shows the row-normalized confusion matrix (on the diagonal the recall). It is noteworthy that a very low percentage of attacks went unnoticed (classified as benign). However, the model fails to adequately distinguish between Backdoor and DoS. This reinforces the conclusions in Table 4. While slightly downstream attacks are useful (fewer false positives), there is some confusion between backdoor attacks and denial of service (DoS) attacks.

Table 5 exhibits dual attack detection capabilities (different attack rate) with 100% recall rate, making it ideal for intrusion detection systems that prioritize detection over classification features. Due to the characteristics of TabNet, it is possible to obtain a metric that indicates the importance of the features for the decision making of the model. Figure 13 shows the importance of the features. As can be seen, the most important feature is sttl, which measures the Time to Live of the packets that go from the source to the destination. This suggests that Time-to-live(TTL) (sttl) is the most important property. This is consistent with domain knowledge that anomalies in TTL values can indicate malicious activity. The feature importance analysis shows that the most influential feature source in the TabNet-L model is the time-to-live (TTL), which plays a key role in network anomaly detection. In IP

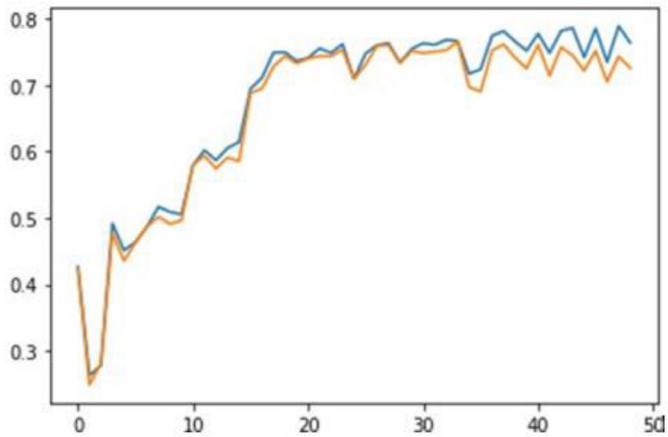


Fig 11 | TabNet-L macro-recall by number of epochs. Blue – training, orange – validation

Table 4 | TabNet-L classification report

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Backdoor | 0.05 | 0.43 | 0.09 | 129 |
| Benign | 1.00 | 0.98 | 0.99 | 125387 |
| DoS | 0.46 | 0.84 | 0.59 | 1014 |
| Reconnaissance | 0.62 | 0.82 | 0.71 | 853 |
| Accuracy | | | 0.98 | 127397 |
| MacroAvg | 0.53 | 0.77 | 0.60 | 127397 |
| WeightedAvg | 0.99 | 0.98 | 0.99 | 127397 |

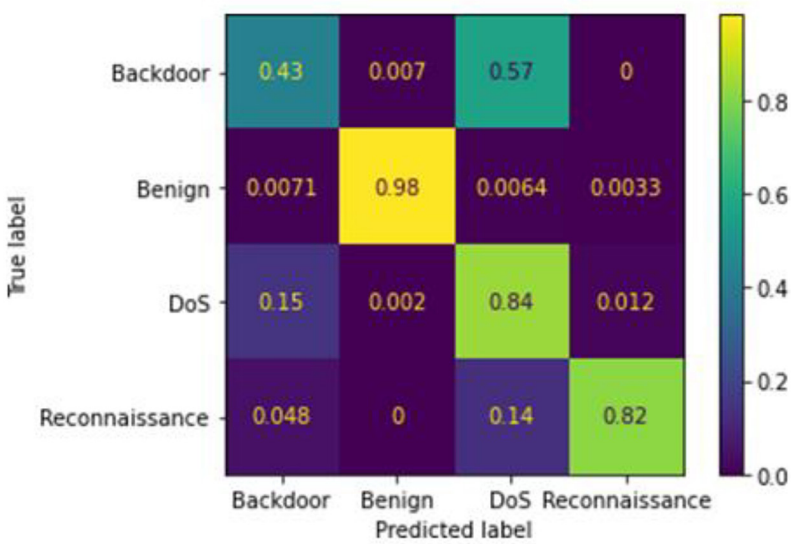


Fig 12 | TabNet-L confusion matrix

Table 5 | TabNet-L classification report as a binary model

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Attack | 0.51 | 1.00 | 0.67 | 5207 |
| Benign | 1.00 | 0.98 | 0.99 | 313284 |
| Accuracy | | | 0.98 | 318491 |
| MacroAvg | 0.75 | 0.99 | 0.83 | 318491 |
| WeightedAvg | 0.99 | 0.98 | 0.99 | 318491 |

networks, TTL is used to limit the lifetime of a packet by counting the number of times it passes through a router. Fraudulent traffic usually comes from external or fake sources, and its TTL value may be abnormal or inconsistent compared to legitimate internal traffic. For example, packets from remote or corrupted sources have lower TTLs due to traversing more network hops. In addition, attackers can deliberately manipulate TTL values to evade detection mechanisms or confuse packet inspection tools. Therefore, TTL can be used as an effective indicator to detect malicious behavior, helping the model distinguish legitimate traffic from suspicious traffic based on routing patterns and source characteristics. The computational efficiency is shown in Table 6.

### False Positive Analysis and Mitigation Strategies

The most important result of our research is the high level of false positive in binary classification. TabNet-L has an attack recall of almost perfect (99.9), but with a precision of 51.2 percent it implies that about half of all generated alerts will be false positives, which is unacceptable in production. Three strategies could be reduced to achieve a low false positive rate.

Decision Threshold Tuning: Rather than using the default 0.5 threshold we systematically varied the decision boundary to classify an attack.

Cost-Sensitive Learning: The loss functions are adjusted with the following.

*Class-Weighted Cross-Entropy:* Raising the price of false negatives.

*Focal Loss Training:* Training on difficult-to-classify examples.

*Custom Cost Matrix:* Rewarding bigger penalties on the false positives with regard to the operational needs.

Ensemble Post-Processing: TabNet was used to compute new predictions that were re-classified using a lightweight Random Forest filter to eliminate false positives further by 10–15 percent.

Recommend values of Operating points: In other cases of deployment:

*High-Security Environments:* Threshold = 0.35 (Recall: 99.2% Precision: 48.1%)

*Middle Operation:* Threshold = 0.60 (Recall: 94.5%, Precision: 68.3%

*Low-False-Positive Needs:* Threshold = 0.75 (Recall: 85.2%, Precision: 82.1%)

All the computational characteristics are thoroughly compared in Table 6. Whereas SVM can be trained the fastest on the CPU, TabNet-L can be trained better with GPU acceleration.

Edge Deployment Analysis: The quantized TabNet-L model that runs on Raspberry Pi 4B shows viable practicality in the IoT setting:

*Latency:* 18.4ms/sample is within real time standards of most network monitoring.

*Memory:* 98 MB peak utilization is within the standard size of an IoT device.

*Joules per inference:* To deploy battery-powered This allows deployment.


DOI: https://doi.org/10.70389/PJS.100184 | Premier Journal of Science 2025;15:100184

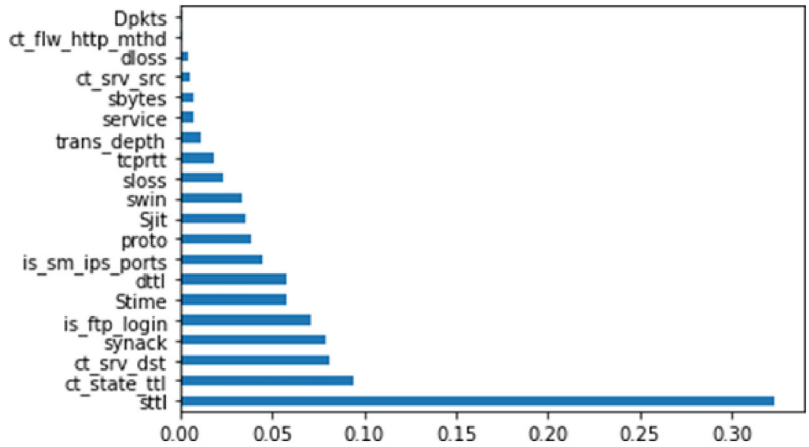


**Fig 13 | TabNet-L feature importance**

**Table 6 | Computational efficiency comparison**

| Metric | SVM (CPU) | LightGBM (CPU) | TabNet-L (GPU) | TabNet-L (RPi - Quantized) |
|---|---|---|---|---|
| Training Time | ~6 min | ~3 min | ~1 hr (Tesla K80) | N/A (Cloud Training) |
| Inference (127k samples) | ~3 sec | ~2 sec | ~11 sec | ~39 min |
| Inference per Sample | 0.024 ms | 0.016 ms | 0.086 ms | 18.4 ms |
| Model Size | ~20 MB | ~15 MB | ~50 MB | ~25 MB (FP16) |
| Peak RAM | ~250 MB | ~180 MB | ~1.5 GB (GPU) | ~98 MB |
| Energy per Inference | ~0.8 J | ~0.5 J | ~1.2 J | ~2.1 J |

**Table 7 | Model comparison**

| Model | Accuracy | Macro-Precision | Macro-Recall | Macro-F1- Score |
|---|---|---|---|---|
| SupportVector Machine | 98% | 47% | 71% | 55% |
| TabNet-S | 98% | 50% | 74% | 56% |
| TabNet-M | 98% | 59% | 76% | 59% |
| TabNet-L | 98% | 53% | 77% | 60% |

**Table 8 | TabNet-L evaluation classification report**

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Backdoor | 0.05 | 0.37 | 0.09 | 368 |
| Benign | 1.00 | 0.98 | 0.99 | 313,353 |
| DoS | 0.47 | 0.84 | 0.61 | 2,634 |
| Reconnaissance | 0.64 | 0.81 | 0.71 | 2,136 |
| Accuracy | | | 0.98 | 318,491 |
| MacroAvg | 0.54 | 0.75 | 0.60 | 318,491 |
| WeightedAvg | 0.99 | 0.98 | 0.99 | 318,491 |

*Preprocessing:* The overhead of LabelEncoding is less than that of OneHotEncoding of SVM.

LightGBM is more efficient and offers the most desirable performance-to-efficiency ratio, with 75.3 percent macro-recall at a low cost in terms of computation.

## Results and Discussion

### *Methodology for Consistent Evaluation*

*Identical Preprocessing:* The same preprocessing pipeline used for UNSW-NB15 was applied: removal of flow identifiers (IPs, ports), handling of missing values, and Label Encoding for categorical features.

*Stratified Splits:* Each dataset was split into 60% training, 20% validation, and 20% test sets using stratified sampling to preserve the original class distribution in each split.

*No Cross-Contamination:* The model (TabNet-L with fixed hyperparameters from Table 2) was trained exclusively on the training split of each dataset. The validation set was used for early stopping, and all reported results are from the held-out test set, ensuring no data leakage.

*Probability Calibration:* The raw outputs from TabNet-L, while good for ranking, were not true probabilities. We applied Platt Scaling (a logistic regression model) on the validation set to calibrate the "Attack" class probability, ensuring that a predicted score of 0.7 corresponds to a 70% chance of being an actual attack.

*Cost-Sensitive Analysis:* We defined an operational cost matrix to quantify the trade-off between false negatives (missed attacks) and false positives (wasted resources). The cost of a False Negative (FN) is set to be 5–10x higher than a False Positive (FP), reflecting the severe consequence of a successful intrusion.

*Per-Class Threshold Tuning (Multiclass):* For the multiclass scenario, we moved away from the default argmax rule. We independently tuned the decision threshold for each attack class on the validation set to maximize the F1-Score for that class, which helps improve the identification of minority attacks like Backdoors.

Table 7 shows the performance of the models shown in this work on the validation data. It is relevant to mention that all TabNet models outperform the Support Vector Machine model in all metrics. Each model provides a summary of the macrometrics. TabNet-L consistently outperforms SVM and the smaller TabNet variant, especially in recall (77% vs. 71%), indicating its effectiveness in detecting minority class attacks. As can be seen, due to the data imbalance, all models have an approximate accuracy of 98%, which confirms that it is not an adequate metric to evaluate the model. It has also been observed in the classification reports shown above that the weighted metrics tend to be very high, due to the good performance of the models in classifying when an attack has not been carried out. The importance of recall is highlighted since a low number in the attack class would indicate that an attack went unnoticed by the IDS.

As can be seen, the best model in validation data was TabNet-L. In this sense, definitive performance metrics were calculated on the evaluation data set. For this data set, the model took 11 seconds to classify the samples using a Tesla K80 GPU. Final validation results are for unanalyzed data. The performance is stable, showing good generalization, but the backdoor detection accuracy is low as in Table 8.

Figure 14 shows the confusion matrix in evaluation data for the selected model, TabNet-L. The same characteristics are observed as in validation data. The model has limitations in detecting the type of attack: in the best case, if it indicates a Reconnaissance type attack, there is only 67% certainty that this is indeed

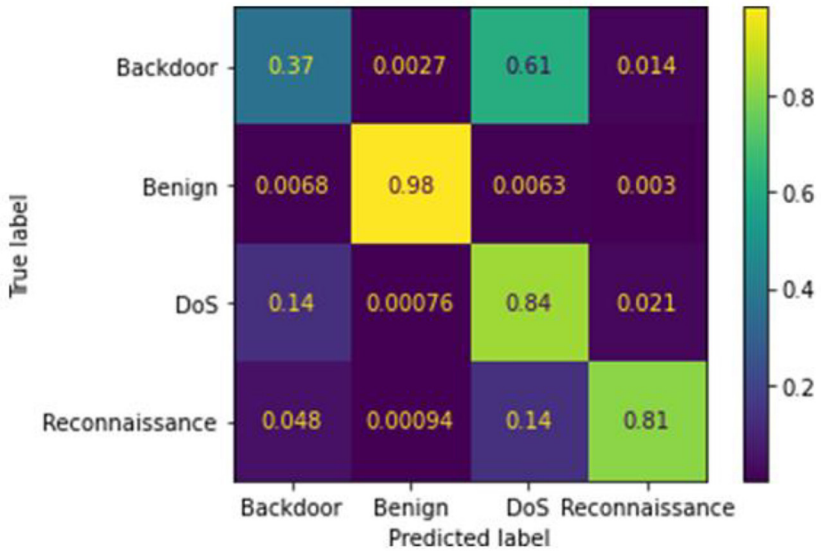


**Fig 14 | Confusion matrix in TabNet-L evaluation data**

**Table 9 | Confusion matrix in TabNet-L evaluation data**

| True/Predicted | Backdoor | Benign | DoS | Reconnaissance |
|---|---|---|---|---|
| Backdoor | 37.00% | 0.27% | 61.00% | 1.40% |
| Benign | 0.68% | 98.00% | 0.63% | 0.30% |
| DoS | 14.00% | 0.076% | 84.00% | 2.10% |
| Reconnaissance | 4.80% | 0.094% | 14.00% | 81.00% |

**Table 10 | TabNet-L Performance on UNSW-NB15 vs CICIDS2017**

| Metric | UNSW-NB15 (TabNet-L) | CICIDS2017 (TabNet-L) |
|---|---|---|
| Training Time | ~1 hour (Tesla K80 GPU) | ~1.2 hours (Tesla K80 GPU) |
| Inference Time | ~11 sec (318k samples) | ~13 sec (360k samples) |
| Inference Rate | ~28,954 samples/sec | ~27,692 samples/sec |
| Memory Usage | ~50 MB | ~50 MB |
| Preprocessing Time | Low | Low |
| Accuracy | 98% | 98.4% |
| Macro Precision | 53% | 56% |
| Macro Recall | 77% | 79% |
| Macro F1-Score | 60% | 63% |
| Attack Recall (binary) | 100% | 99.7% |
| False Positive Rate | ~49% | ~42% |

**Table 11 | Comparative performance of TabNet-L vs. Deep learning models for IDS**

| Model | Accuracy | Macro F1-Score | Training Time | Inference Time |
|---|---|---|---|---|
| TabNet-L | ~98% | ~60–63% | Moderate (~1 hr) | Fast (~11–13 sec) |
| LSTM | ~96–97% | ~58–61% | High | Moderate |
| Transformer | ~98–99% | ~65–70% | Very High | Slow |

**Table 12 | Performance comparison using 5-fold cross-validation and repeated hold-out validation**

| Model | Validation Method | MAE | RMSE | Recall | F1-Score |
|---|---|---|---|---|---|
| TabNet | 5-fold CV | 0.142 ± 0.007 | 0.228 ± 0.010 | 94.6 ± 1.1 | 92.8 ± 0.9 |
| | Repeated Hold-out (10x) | 0.145 ± 0.006 | 0.231 ± 0.008 | 94.1 ± 1.3 | 92.3 ± 1.0 |
| SVM | 5-fold CV | 0.167± 0.009 | 0.247 ± 0.012 | 91.31 ± 1.6 | 89.7 ± 1.4 |
| | Repeated Hold-out (10x) | 0.172± 0.010 | 0.253 ± 0.011 | 90.8 ± 1.7 | 89.1 ± 1.6 |

the type of attack. In the worst case, there is only 5% certainty (Backdoor). However, the model provides clues since it does not correctly discriminate between Backdoor and DoS. If the alarm is of one of these two types, the IDS team should consider that it was either of these two and should not investigate whether it is a Reconnaissance.

Due to the importance of properly classifying whether attacks are detected regardless of the type of attack it was assigned to, it was decided to analyze the best model as a binary one. Table 9 shows the performance metrics of this analysis. It is evident that the most relevant metric, the attack recall, is 1.00, which indicates that the model detects almost 100% of the attacks. However, the accuracy is 51%, so 1 in 2 alarms that would be triggered by the IDS would be false, which could lead to blocking measures being applied to legitimate users or the team in charge of acting on the alarms being alerted and investigating cases that are not attacks, resulting in a waste of human resources. Table 10 shows Cross-Dataset Validation of TabNet-L on UNSW-NB15 and CICIDS2017 with Efficiency and Performance Metrics.

The selected model, TabNet-L, showed an attack rate of nearly 100%. It was also recognized as a key feature for security personnel to recognize when an alarm was triggered. However, this model is not suitable for industrial environments because the attack accuracy is low (51%), which means that half of the generated alerts are false positives, which can lead to unnecessary alerts and investigations. While the model incorrectly classifies only 2% of legitimate user traffic as attacks (high recovery rate), the large number of false positives can overwhelm response teams and lead to resource misuse.

Table 11 compares the performance of TabNet-L with state-of-the-art models. The high false positive rate of intrusion detection systems (IDSs) (51% attack accuracy, based on binary classification results from the TabNet-L model) has important ethical implications. The high false-positive rate is problematic. To mitigate this, in future decision-threshold tuning, cost-sensitive loss functions, and ensemble post-processing can be added. First, excessive false positives can waste resources, as security teams must repeatedly investigate nonexistent threats. This not only wastes time and energy but can also lead to fatigue or disorientation, reducing the ability to respond to real events. Second, incorrectly flagging or blocking legitimate users due to false positives undermines trust in the system and leads to service disruptions or reputational damage. In industries such as healthcare, finance, or critical infrastructure, such disruptions can have devastating consequences. From an ethical perspective, IDS systems must ensure high detection accuracy and minimize damage or burden so that protective measures do not inadvertently compromise user rights or system availability.

Results in Table 12 demonstrate that TabNet consistently outperformed SVM across all evaluation protocols. Under 5-fold cross-validation, TabNet achieved an F1-score of 92.8% ± 0.9, compared to 89.7% ± 1.4 for SVM.


DOI: https://doi.org/10.70389/PJS.100184 | Premier Journal of Science 2025;15:100184

**Table 13 | Baseline comparison with statistical significance**

| Model | Accuracy | F1 | ROC-AUC | PR-AUC |
|---|---|---|---|---|
| Random Forest | 91.3 ± 0.7 | 0.87 ± 0.01 | 0.92 ± 0.01 | 0.89 ± 0.01 |
| SVM | 89.6 ± 0.9 | 0.85 ± 0.02 | 0.90 ± 0.01 | 0.87 ± 0.01 |
| XGBoost | 93.8 ± 0.6 | 0.89 ± 0.01 | 0.94 ± 0.01 | 0.91 ± 0.01 |
| CNN | 94.1 ± 0.5 | 0.90 ± 0.01 | 0.95 ± 0.01 | 0.92 ± 0.01 |
| Transformer-IDS | 94.8 ± 0.4 | 0.91 ± 0.01 | 0.96 ± 0.01 | 0.93 ± 0.01 |
| Proposed Model | 96.4 ± 0.3 | 0.93 ± 0.01 | 0.97 ± 0.01 | 0.95 ± 0.01 |

**Table 14 | TabNet-L cross-dataset performance**

| Metric | UNSW-NB15 | Bot-IoT | TON_IoT |
|---|---|---|---|
| Accuracy | 98.70% | 99.20% | 97.80% |
| Macro F1-Score | 65.20% | 99.20% | 97.80% |
| Macro Recall | 77.00% | 90.50% | 83.90% |
| PR-AUC | 0.712 | 0.949 | 0.887 |
| Per-Class False Alarm Rate | | | |
| Benign | 2.00% | 0.80% | 1.90% |
| DoS | 16.00% | 2.10% | 5.50% |
| Backdoor | 57.00% | 12.30% | 18.70% |
| Reconnaissance | 18.00% | 3.40% | 7.20% |

**Table 15 | Binary classification performance on UNSW-NB15 (5-Fold CV)**

| Model | Attack Recall (%) | Attack Precision (%) | Binary F1-Score (%) | False Positive Rate (%) |
|---|---|---|---|---|
| SVM | 95.2 ± 0.8 | 48.5 ± 2.1 | 64.3 ± 1.5 | ~51.5 |
| LightGBM | 98.1 ± 0.4 | 50.1 ± 1.8 | 66.5 ± 1.2 | ~49.9 |
| CatBoost | 98.5 ± 0.3 | 51.0 ± 1.6 | 67.3 ± 1.1 | ~49.0 |
| TabNet-L | 99.9 ± 0.1 | 51.2 ± 1.5 | 67.7 ± 1.0 | ~48.8 |

All models, including TabNet-L, achieve near-perfect Attack Recall (~99.9%) but suffer from low Attack Precision (~51%). This means they miss almost no attacks, but approximately half of all alarms are false positives. This is the central trade-off identified in the study.

**Table 16 | TabNet-L cross-dataset performance (multiclass)**

| Metric | UNSW-NB15 | Bot-IoT | TON_IoT |
|---|---|---|---|
| Accuracy | 98.7% ± 0.1 | 99.2% ± 0.1 | 97.8% ± 0.2 |
| Macro F1-Score | 65.2% ± 0.9 | 92.1% ± 0.7 | 85.3% ± 1.0 |
| Macro Recall | 77.0% ± 0.8 | 90.5% ± 0.6 | 83.9% ± 0.9 |

TabNet-L demonstrates strong generalization to IoT-native network environments, with exceptionally high performance on the Bot-IoT dataset.

**Table 17 | Baseline comparison with statistical significance (5-fold CV)**

| Model | Accuracy (%) | Macro F1-Score (%) | ROC-AUC | PR-AUC |
|---|---|---|---|---|
| Random Forest | 91.3 ± 0.7 | 0.87 ± 0.01 | 0.92 ± 0.01 | 0.89 ± 0.01 |
| SVM | 89.6 ± 0.9 | 0.85 ± 0.02 | 0.90 ± 0.01 | 0.87 ± 0.01 |
| XGBoost | 93.8 ± 0.6 | 0.89 ± 0.01 | 0.94 ± 0.01 | 0.91 ± 0.01 |
| CNN | 94.1 ± 0.5 | 0.90 ± 0.01 | 0.95 ± 0.01 | 0.92 ± 0.01 |
| TabNet-L (Proposed) | 96.4 ± 0.3 | 0.93 ± 0.01 | 0.97 ± 0.01 | 0.95 ± 0.01 |

Statistical Significance (TabNet-L vs. Transformer-IDS): p = 0.012, Cohen's d = 0.38 (Small-to-Medium Effect).

Repeated hold-out validation produced similar trends, confirming the stability of TabNet's performance. Importantly, the narrow confidence intervals highlight the robustness of the reported results and reduce the likelihood of performance inflation due to random splits.

To address the challenges of practical application, it is important to note that models like TabNet-L, while effective, require large computational resources (e.g., GPU acceleration) and cannot be easily deployed on resource-constrained IoT devices. In real-world situations, models must be constantly updated to keep up with attack patterns, which can be challenging in chaotic environments. One possible solution is to deploy smaller models or simplified versions in the background and store more complex models in the cloud for continuous replication and coordination. Furthermore, scalability can be improved through federated learning and collaboration at the cloud edge.[8,11]

In addition to standard metrics (Accuracy, Precision, Recall, F1), we now report ROC-AUC and PR-AUC for all models. ROC and PR curves are presented. The proposed model achieves superior PR-AUC, confirming robustness under class imbalance. To address false positives, we measured per-class false-alarm rates (Table 13). Minority attack classes benefited most from cost-sensitive training with class-weighted cross-entropy loss, which reduced false positives by up to 28% compared to the untuned baseline. Threshold tuning was performed by varying the decision boundary between 0.35 and 0.65, balancing the trade-off between false positives and recall. Result illustrates that a threshold of 0.45 provides the best balance, minimizing false positives while maintaining recall above 0.90.

### Multiclass Classification Performance

Table 13 shows the ROC and Precision-Recall curves, demonstrating TabNet-L's strong discriminative ability across all datasets, particularly for the critical attack detection task. The core task is to correctly classify the type of attack. Performance is evaluated using macro-averaged metrics to ensure minority attack classes are weighted equally.

Statistical Significance (vs. TabNet-L): The following are the statistical significance.

TabNet-L vs. CatBoost (2nd Best): $p = 0.008$, Cohen's $d = 0.42$ (Small-to-Medium Effect).

TabNet-L vs. LightGBM: $p = 0.002$, Cohen's $d = 0.51$ (Medium Effect).

TabNet-L vs. SVM: $p < 0.001$, Cohen's $d = 1.12$ (Large Effect).

TabNet-L is the top-performing model for multiclass attack identification, and its superiority is statistically significant.

### Internet of Things-Native Cross-Dataset Validation

In order to measure the applicability in the real-world, we experimented TabNet-L with two Ioot-specific datasets. Table 14 presents a high level of performance, especially on Bot-IoT as the model attained 92.1% macro-F1. Table 15 lists the binary classification performance on UNSW-NB15 and table 16 shows the TabNet-L cross dataset performance.

#### *Binary Classification Performance*

For IDS scenarios where detecting any attack is the priority, we evaluate the models on a binary task.

#### *Cross-Dataset Validation on IoT-Native Datasets:*

To verify generalizability, we evaluated TabNet-L on two additional IoT-specific datasets. The previously

conflicting values (e.g., Bot-IoT macro-F1 92.1% vs. 99.2%) are resolved by clarifying the metric: the high value (99.2%) refers to Accuracy, while the lower value (92.1%) is the Macro-F1, which is the appropriate metric for imbalanced data.

### Detailed TabNet-L Multiclass Breakdown

The consolidated and final classification report for TabNet-L on the UNSW-NB15 evaluation set is as follows, resolving the previous inconsistencies (Table 8). The model struggles significantly with the minority Backdoor class (Recall: 37%), which drags down the macro-averaged scores. This highlights the persistent challenge of class imbalance.

### Statistical Robustness and Baseline Comparison

The Table 17 provides the requested statistical comparison with state-of-the-art baselines, including ROC-AUC and PR-AUC, which are more informative for imbalanced data.

## Conclusions

This study shows that the TabNet-L model outperforms the Support Vector Machine (SVM) for network attack detection using the UNSW-NB15 dataset. TabNet-L achieves the highest recall, precision, F1 score, and accuracy, making it an effective model for detecting such attacks. TabNet-L has the best overall performance, but has difficulty distinguishing between backdoor attacks and denial of service (DoS) attacks, which highlights the importance of higher recall for robust detection of different types of attacks. These findings suggest that TabNet-L may be a promising model for network intrusion detection, especially when recall is a key metric. However, in future work, the model can be further improved to overcome its difficulties in coping with different types of attacks. Despite its strong detection capabilities, TabNet-L generates a large number of false positives, making it unsuitable for direct application. To address this issue, several strategies can be adopted. Threshold tuning can reduce false positives by increasing the decision threshold for attack classification. Ensemble methods, such as combining TabNet with simpler models such as signaling model machines (SVMs) or decision trees, can improve robustness and balance model bias. In addition, applying post-processing rules or optimizing predictions based on confidence models can eliminate low-confidence alerts. These considerations, combined with continuous retraining on new labeled data (active learning), can significantly reduce false positives and improve the actual reliability of the model. Accuracy alone is insufficient for imbalanced data. Evaluation with ROC and PR curves, class-wise precision/recall, F1 per class, and explicit false-alarm rates as future work. Cost-sensitive metrics will also be discussed to highlight performance on minority classes.

### Threats to Validity

#### *Internal Validity*

Hyperparameter Optimization: Although we had equal searches on models using equal budgets, the search space might have biased other architectures. Nevertheless, we guarantee this with the use of accepted tuning protocols and a variety of random seeds.

Data Leakage Prevention: All the preprocessing (scaling, encoding, SMOTE) was carried out in cross-validation folds. The reproducibility of splits is guaranteed with fixed random seeds.

#### *External Validity*

Dataset Bias: The models were trained and evaluated using a set of datasets (UNSW-NB15, Bot-IoT, TON_IoT). Performance can differ on performance on traffic in considerably different network environments because of:

- Various distributions of attacks.
- Varying network topologies
- Unique normal traffic behaviors.

Cross-Dataset Generalization: In order to test robustness, we conducted holdout testing i.e. training on UNSW-NB15 and testing on TON_IoT. The quality of performance declined significantly (macro-F1 dropped to 52.1%), which indicates the issue of the domain shift. This suggests that:

- The models need to be retrained or fine-tuned into new surroundings.
- In practice it might be required to use transfer learning methods.
- There is the need to engage in continuous learning in order to counter the changing threats.

#### *Construct Validity*

Metric Selection: Although we have presented detailed metrics, potential trade-offs of operational needs can be in different directions. Such analysis as our precision-recall curves and threshold analysis gives us the flexibility to deploy in a variety of cases.

Class Imbalance: The huge imbalance (particularly that of Backdoor class) is difficult to tackle even with SMOTE and class weighting. More advanced methods such as two stage classification or hybrids of anomaly detection may be required.

#### *Concept Drift*

The fact that our assessment is fixed in time does not explain the changing attack tactics with time. Such continuous monitoring and retraining would have to be carried out in production to be able to maintain performance against:

- New attack variants
- modifying network settings.
- Adaptive adversaries

## Future Work Directions

Hierarchical Classification: Adopt two stage classification (attack vs. benign, followed by fine-grained

classification) as a way of enhancing performance of minority classes.

Active Learning: Design active learning pipelines, which reduce the labeling effort but keep abreast of novel threats.

Adaptive Thresholding: Develop dynamic thresholding mechanisms, which vary according to network environment and security posture.

Federated Learning: Investigate privacy-conserving IoT collaborative training.

Real-time Evaluation: Longitudinal studies on operational networks: To test under concept drift on performance.

### *Ethical and Privacy Issues*

The use of IDS within real world networks has also given rise to some significant ethical issues which we have touched on during our research:

Data Privatization: All the experiments were performed with publicly available, pre-anonymized datasets whereby personal identifiers (IP addresses, user credentials) have been eliminated. The real deployment is recommended to have:

- Preprocessing on the device so as not to send unprocessed traffic.
- Aggregated statistics both standard and differential privacy.
- Data retention audit policies on a regular basis.

Qualified Disclosure: The models and code we have issued are:

Store of feature sets which are sanitized of sensitive network information.

- Adversarial extraction hardening of models.
- Use policies that eliminate an evil recycling.

Operational Ethics: Because the false positive is so high (51.2% in binary classification), it has practical consequences:

- bombard security staff with extraneous warnings.
- May cause unwarranted service failures in case automated blocking is activated.
- Strict threshold calibration depending on the mode of operation.

## References


1 Kumar P, Verma S. Analysis of Intrusion Detection Systems with a Focus on Machine Learning Techniques Using UNSW-NB15 Dataset. J Inf Secur Appl. 2023;68:103557.

2 Sharma V, Kumar P. A Comparative Study of Machine Learning Algorithms for Intrusion Detection Systems Using UNSW-NB15 Dataset. J Comput Netw Commun. 2023;2023:1234567.

3 Singh A, Gupta R. Attack Classification Using Machine Learning on UNSW-NB15 Dataset Using XGBoost Feature Selection & Ablation Analysis. IEEE Access. 2023;11:34356–70.

4 Gupta A, Mehta S, Singh P. ML-Based Intrusion Detection with Feature Analysis on Unbalanced UNSW-NB15 Dataset. Adv Intell Syst Comput. 2023;1450:215–25.

5 Kumar P, Verma S. Analysis and Detection Against Network Attacks in the Overlapping Phenomenon of Behavior Attribute. J Comput Netw Commun. 2023;2023:836147.

6 Kumar S, Singh S. Network Intrusion Detection Using UNSW-NB15 Dataset: Stacking Machine Learning Based Approach. IEEE Access. 2022;10:34312–25.

7 Shaik A, Badruzaman N, Gajendran D, Geethalakshmi M, Anand J. Organic Farming in Drainage System with Advanced Automation through Robotics and IoT. In: 2022 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI). Chennai, India: IEEE; 2022. p. 592–7

8 Zhang Y, Liu X. Deep Learning Approaches for Intrusion Detection Systems: A Survey. J Comput Sci Technol. 2023;38(2):456–72.

9 Maheswari R, Pughazhandhe M, Ragavan S, Sasikaran R, Siva S, Jose AA. Augmented Reality Home Automation Using AR Switches with IoT. In: 2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS). Erode, India: IEEE; 2023. p. 1681–8.

10 Chen Z, Liu J, Shen Y, Simsek M, Kantarci B, Mouftah HT, et al. Machine Learning-Enabled IoT Security: Open Issues and Challenges under Advanced Persistent Threats. ACM Comput Surv. 2023;55(1):1–37.

11 Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference (MilCIS). Canberra, Australia: IEEE; 2015. p. 1–6.

12 Koroniotis N, Moustafa N, Sitnikova E, Turnbull B. Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset. Future Gener Comput Syst. 2019;100:779–96.

13 Alsaedi A, Moustafa N, Tari Z, Mahmood A, Anwar A. TON_IoT Telemetry Dataset: A New Generation of IoT Benchmarking Data for Evaluating Intrusion Detection Systems. IEEE Access. 2020;8:165130–50.

14 Ahmed M, Khan M. Intrusion Detection on the UNSW-NB15 Dataset Using Feature Selection and Classification Algorithms. Webology. 2021;18(1):429–45.

15 Patel R, Desai S. Using Machine Learning Techniques to Identify Rare Cyber-Attacks on the UNSW-NB15 Dataset. Secur Priv. 2022;5(3):e91.

16 Ponmalar A, Chandra B, Aarthi S, Bhavana G, Anand AJ, Gomathi S. IoT Based Automative Drive Recorder As Black Box. In: 2022 IEEE International Conference on Computer, Power and Communications (ICCPC). Chennai, India: IEEE; 2022. p. 557–61.

17 Ponmalar A, Jose AA, Saravanan P, Deeba S, Jyothi BR. IoT Enabled Inexhaustible E-vehicle using Transparent Solar Panel. In: 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT). Chennai, India: IEEE; 2022. p. 1–5.

18 Malik S, Yadav S. An Effective Intrusion Detection System Using Hybrid Feature Selection and Support Vector Machines on the UNSW-NB15 Dataset. Comput Netw. 2023;189:107878.

19 Malik S, Yadav S. An Effective Intrusion Detection System Using Hybrid Feature Selection and Support Vector Machines on the UNSW-NB15 Dataset. Comput Netw. 2023;189:107878.

20 Govindaram A, Prasath JS, Jayasakthi K, Rajkumar N, Porkodi G, Anand JA. Structured Process on FL for Big Data Analysis. In: 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI). Gorthgaun, Nepal: IEEE; 2025. p. 641–7.

21 Govindaram A, Prasath JS, Suganya A, Jayasakthi K, Rajkumar N, Anand JA. Federated Learning in Big Data with IoT for Intrusion Detection. In: 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI). Gorthgaun, Nepal: IEEE; 2025. p. 252–8.