JOURNAL OF Emerging Paradigms in Computing Systems JEPCS

Journal of Emerging Paradigms in Computing Systems (JEPCS) | Volume 01, Issue No. 02, 2025

ARTICLE TYPE: Research Article

Article ID: JEPCS-2025-0102-0043

A Comparative Study of Classical and Deep Learning Models for Edge-Oriented Network Anomaly Detection

Ayesha Iftikhar[1,*], **Sadia Aslam**[2]

[1] Lahore Business School, The University of Lahore, Lahore, Pakistan
[2] Faculty of Information Technology & Computer Science, University of Central Punjab, Lahore, Pakistan

*Corresponding author: aiftikhar184@gmail.com
Contributing authors: aiftikhar184@gmail.com, sadia.aslam@ucp.edu.pk

Received: March 20, 2025 **Accepted:** June 28, 2025

Abstract- Detection quality and model cost cannot be separated since an edge-side anomaly detector requires less memory and processing power than a cloud-hosted detector. Under a single binary intrusion-detection technique, we evaluate logistic regression, a linear support vector machine, and a compact deep neural network. The entire dataset is used for final evaluation, whereas the standardized 20% NSL-KDD training subset is used for development. Categorical variables are one-hot encoded, numerical variables are standardized from training statistics, and decision thresholds are chosen on a held-out validation partition. We test three input sizes: all 118 encoded features, the 20 highest-ranked features, and the top 10 features according to a Fisher score computed on training data only. With the full feature set, the compact neural network obtained an F1-score of 0.7591 and an ROC-AUC of 0.8799. The 20-feature conditions changed the result. Logistic regression reached 0.7998 accuracy, 0.8054 F1-score, 0.8113 balanced accuracy, and 0.6186 Matthews correlation coefficient with 21 parameters and an estimated 0.082 KB of FP32 parameter storage. The 20-feature neural model gave the highest ROC-AUC, 0.9230, with 1,217 parameters. Using only 10 features lowered the false-alarm rate for the linear models, but attack recall also fell. On this benchmark, feature budget altered both detection behavior and model footprint. Edge-oriented model selection should therefore account for threshold-dependent errors and computational cost in addition to aggregate classification scores.

Keywords- Network anomaly detection; intrusion detection system; edge computing; machine learning; deep learning

1. Introduction

An intrusion detection system (IDS) monitors traffic or host activity for patterns associated with malicious behavior. In anomaly-based IDS research, machine-learning models are trained from labeled traffic so that detection is not limited to a manually maintained signature set [1, 2]. The resulting classifier still has to operate somewhere in the network. For an edge deployment, the practical question is therefore not only whether the model detects attacks, but also whether the required computation is reasonable at the point where the traffic is observed. Edge computing moves part of the processing path closer to the source of the data [3]. In IoT and fog systems, this can reduce the amount of traffic sent to a remote cloud and can shorten the response path, although an edge node usually has fewer compute and storage resources than a cloud server [4]. It is not sufficient to evaluate an IDS operating at the edge just on its ability to distinguish between legitimate and malicious traffic. While the memory and processing needed for inference must remain within the bounds of the edge node, its false-alarm behavior must continue to be acceptable for the intended usage. Even if a detector has good statistical performance, it could not be appropriate for the target platform if its execution costs are too high. Neural and classical models take distinct approaches to this issue. Neural networks are able to capture nonlinear

connections among the input variables, whereas linear classifiers and support vector machines employ relatively simple decision functions [5,6]. IoT-oriented intrusion detection has also examined autoencoders and lightweight support vector machines [7,8]. The model family alone cannot be used to interpret their performance. The distribution of the test traffic, the amount of retained characteristics, the chosen decision threshold, and the representation given to the classifier can all have an impact on the observed outcome. A portion of the edge inference cost can be reduced by reducing the input space. Less telemetry has to be gathered and analyzed when fewer traffic fields are needed. The quantity of inputs, as well as well as well as well as well. Ambusaidi et al. [9] reported that informative feature subsets might eliminate superfluous dimensions without sacrificing effective intrusion-detection capabilities. However, when the feature set is decreased, some information is unavoidably lost, which might change the ratio of recall to precision. Because of this, accuracy alone should not be used to evaluate the impact of feature reduction. What may be logically inferred from a comparison is also influenced by the dataset selection. UNSW-NB15, Bot-IoT, TON_IoT, Edge-IIoTset, and CICIoT2023 were built for more current network or IoT environments [10–14]. The current IoT protocols and the variety of network attacks are not replicated by NSL-KDD, which is significantly older. It is used in this study for a different reason. The dataset was created to reduce the duplicate-record bias related to KDD Cup 99 [15], and its development and official test files offer a fixed benchmark. Therefore, rather than being proof of present IoT threat coverage, the results are taken as evidence concerning model behavior under a controlled feature-budget experiment.

In this paper, two linear classifiers are compared with a compact neural network and the evaluation method is kept the same. All models are trained and tested using the same preprocessing process, validation partition and official NSL-KDD test set. The input representation is then manipulated so that the impact of feature budget on the results can be studied isolated from the changes in the evaluation protocol. Feature ranking is performed only on the training partition. We report accuracy and F1-score together with balanced accuracy, Matthews correlation coefficient (MCC), false-alarm rate (FAR), and receiver operating characteristic area under the curve (ROC-AUC). Parameter count, estimated FP32 storage, and a structural multiplication count are included as reproducible proxies for edge-side model cost.

The paper has four specific contributions.

- Linear regression, a linear support vector machine (SVM), and a compact deep neural network (DNN) are evaluated under one binary anomaly-detection protocol.
- The encoded representation is then reduced from 118 features to 20 and 10 features with a Fisher ranking fitted on training data only.
- Detection results are reported together with model size and per-sample structural computation.
- Finally, the analysis separates threshold-independent discrimination from thresholded behavior, which allows the ROC-AUC result to be interpreted alongside F1-score and FAR rather than as a substitute for them.

The organization of the paper is as follow: Section 2 reviews the IDS and edge-security work most closely related to this comparison. Section 3 defines the dataset, preprocessing, feature ranking, classifiers, and evaluation measures, and Section 4 records the experimental scenario. Section 5 reports the results and examines the detection-resource trade-offs. Limitations are discussed in Section 6, followed by the conclusion and future work in Section 7.

2. Related Work

Earlier IDS research covers supervised and unsupervised learning, neural networks, and hybrid methods, as summarized by Buczak and Guven [1] and Liu and Lang [2]. A recurring concern is whether a result obtained from a closed benchmark transfers to an operational network. Sommer and Paxson [16] described this mismatch between experimental evaluation and open-world deployment. We address only one part of that problem by keeping an external official test set separate from the development split, rather than forming train and test data by randomly partitioning one pooled sample. Deep learning has been studied both as a detector and as a way to learn features for a conventional classifier. Diro and Chilamkurti [5] compared distributed and centralized deep-learning detection in an IoT scenario. Vinayakumar et al. [6] evaluated deep neural architectures across several intrusion-detection benchmarks. Al-Qatf et al. [17] instead used a sparse autoencoder to generate features for an SVM. These studies motivate the neural baseline used here, but they do not imply that additional model capacity is always useful on a constrained execution node. Resource constraints are more explicit in IoT and edge-oriented IDS work. Chaabouni et al. [18] review the difficulty of applying conventional learning-based IDS designs to heterogeneous IoT devices. Jan et al. [8] used a lightweight SVM with a small feature set, whereas Meidan et al. [7] modeled IoT device behavior with deep autoencoders for Mirai and BASHLITE detection. Bangui and Buhnova [19] targeted edge networks with a distributed deep-forest design and feature optimization. Mahadevappa et al. [20] used linear discriminant analysis with logistic regression in an edge-security model. Together, these works motivate evaluating detection behavior and computational footprint in the same experiment. Benchmark design has changed substantially since NSL-KDD. NSL-KDD was introduced to reduce redundancy and evaluation bias present in KDD Cup 99 [15]. UNSW-NB15 includes newer benign and digital malicious traffic [10], while CICIDS2017 uses benign user profiles and several attack scenarios [21]. Bot-IoT focuses on botnet activity in an IoT-oriented testbed [11]. TON_IoT adds network, telemetry, and operating-system data [12]; Edge-IIoTset uses an IoT/IIoT environment with edge and cloud components [13]; and CICIoT2023 provides a large-scale IoT attack benchmark [14]. For claims about current IoT attack

realism, the newer datasets are more appropriate than NSL-KDD. Our use of NSL-KDD is narrower. It provides a fixed benchmark for examining what happens when classifier complexity and feature budget are varied under the same protocol. A model that is small enough for edge-side inference would still need to be tested on traffic from the intended deployment domain before any operational claim could be made. Feature selection is also relevant to deployment cost. Ambusaidi et al. [9] used a filter-based method to remove unnecessary inputs, and Jan et al. [8] worked with a small feature set in a lightweight IoT detector. We therefore vary feature budget explicitly rather than fixing it as a preprocessing choice. Recall, false alarms, ROC-AUC, parameter count, and structural computation are then examined for each budget. Table 1 brings together the studies used to frame the comparison in this paper. The table is intended to show how earlier work relates to the present experiment rather than to rank the studies. In our evaluation, the data split and preprocessing procedure remain unchanged across models. What changes are the classifier family and the number of retained features, allowing detection results to be examined alongside simple measures of model footprint.

Table 1: *Representative IDS studies used to frame the model and resource comparison.*

Study	Scenario / data	Main modeling focus	Relation to the present study
Diro and Chilamkurti [5]	IoT / distributed detection	Deep learning	Examines distributed and centralized detection, whereas feature-budget variation is outside its main evaluation focus.
Jan et al. [8]	IoT	Lightweight SVM	Shows how a reduced input set can be used in a lightweight IoT intrusion detector.
Vinayakumar et al. [6]	Multiple IDS benchmarks	Deep neural networks	Studies neural intrusion-detection models across several benchmarks, with model footprint receiving less attention than detection performance.
Bangui and Buhnova [19]	Edge computing	Deep forest and optimization	Addresses lightweight intrusion detection for edge-computing environments.
Mahadevappa et al. [20]	Edge computing	LDA and logistic regression	Applies classical learning methods within an edge-security architecture.
This study	NSL-KDD official test set	LR, linear SVM, compact DNN	Uses one fixed preprocessing and evaluation procedure while comparing three model families under 118-, 20-, and 10-feature inputs.

3. Materials and Methods

3.1. Study Design

The experiment compares each NSL-KDD record with attack traffic and each NSL-KDD record with normal traffic, and tries to solve the problem as a binary classification. Logistic regression (LR), a linear support vector machine (SVM), and a compact feed-forward deep neural network (DNN) are considered as classifiers. The same kind of encoded input fields are used for all models per feature-budget condition, and so any differences in the results is not due to the different representations used. Construction and evaluation of the models are clearly separated in the stages. The development partition is used for feature ranking and model fitting, and the validation partition is used for threshold selection. The official test set is used only for the final evaluation, and is not used for the parameter estimation, feature selection or threshold selection. The processing flow is shown in Figure 1. The 20% NSL-KDD development file is divided into two parts: training files and testing files. The training partition provides preprocessing statistics, feature ranking and model parameters. Thresholds are selected from the validation data. The models are then tested on the entire official NSL-KDD test file after these steps are correct.

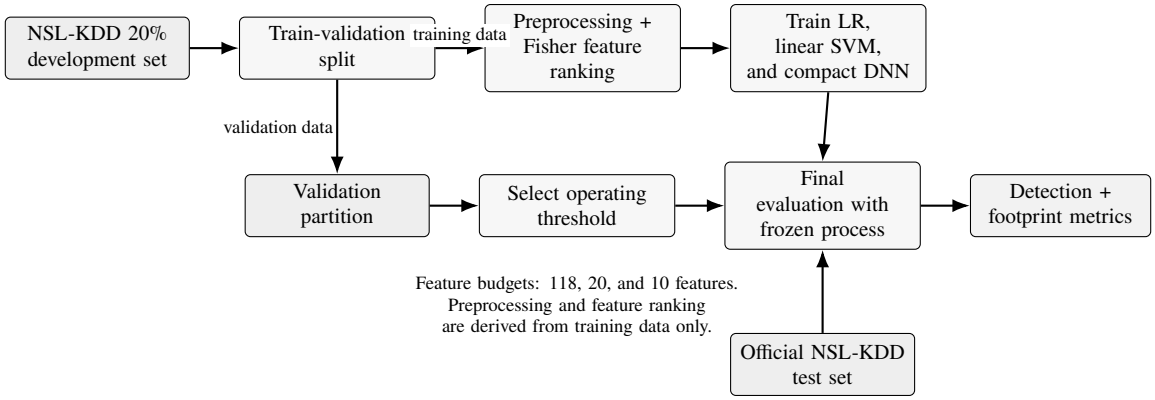


Figure 1: *Experimental workflow for model development and evaluation.*

3.2. NSL-KDD Data and Evaluation Split

NSL-KDD was proposed as a revised form of KDD Cup 99 with fewer duplicate records and a different difficulty distribution [15]. A record contains 41 traffic attributes, an attack-type label, and a difficulty field. We use KDDTrain+_20Percent.txt for development and reserve KDDTest+.txt for the final evaluation. The difficulty field is

excluded from the predictors. The development file contains 25,192 records, comprising 13,449 normal and 11,743 attack records. A 80/20 split assigns 20,153 records to training and 5,039 to validation. The untouched official test file contains 22,544 records: 9,711 normal and 12,833 attacks. Table 2 gives the partition counts.

Table 2: *Dataset partitions used in the experiment.*

Partition	Normal	Attack	Total
Training	10,759	9,394	20,153
Validation	2,690	2,349	5,039
Official test	9,711	12,833	22,544

We do not treat NSL-KDD as a current IoT traffic trace. Here it is a reproducible benchmark with a fixed external test distribution. Section 2 describes newer datasets, and the limitation is considered again later in the paper. Any model intended for a particular edge network would need a separate evaluation on traffic collected from that scenario.

3.3. Preprocessing and Leakage Control

Among the 41 original traffic attributes, protocol type, service, and connection flag are categorical. Their vocabularies are taken from the development data and converted to one-hot indicators, which yields 80 binary columns. The other 38 attributes are numerical. The full encoded representation therefore contains 118 features.

For numerical feature x_j, only the training partition is used to estimate the mean μ_j and standard deviation σ_j. We then compute

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}. \tag{1}$$

Validation and test records are transformed with these training-derived statistics. The service value tftp_u appears in the official test file but not in the development vocabulary. We encode its service indicators as all zeros rather than extending the vocabulary from test data. The experiment does not use oversampling, undersampling, or digital examples.

3.4. Feature-Budget Construction

Three input budgets are evaluated: the full 118-feature representation, the top 20 features, and the top 10 features. Ranking is fitted only on the training partition. For feature j, we use the binary Fisher score

$$F_j = \frac{(\mu_{j,1} - \mu_{j,0})^2}{\sigma_{j,1}^2 + \sigma_{j,0}^2 + \epsilon}, \tag{2}$$

where class 1 is the attack class, class 0 is the normal class, and ϵ avoids division by zero. Under this univariate measure, a larger value means that the two training classes are more separated along feature j.

Table 3 lists the 20 highest-ranked encoded fields. Connection-state and service-rate variables occupy many of the top positions, and two one-hot flag indicators are also selected. Because the ranking is computed independently of the classifier, the same reduced input is supplied to LR, SVM, and DNN within each budget.

Table 3: *Top 20 encoded features selected by the training-only Fisher score.*

Rank	Feature	Rank	Feature
1	flag=SF	11	service=http
2	same_srv_rate	12	count
3	dst_host_srv_count	13	service=private
4	logged_in	14	dst_host_count
5	dst_host_same_srv_rate	15	service=domain_u
6	dst_host_srv_serror_rate	16	rerror_rate
7	dst_host_serror_rate	17	dst_host_srv_rerror_rate
8	flag=S0	18	srv_rerror_rate
9	serror_rate	19	dst_host_rerror_rate
10	srv_serror_rate	20	dst_host_diff_srv_rate

3.5. Classifiers

Logistic regression provides the first classical baseline. For encoded input vector $\mathbf{x}$, the attack probability is

$$p(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp[-(\mathbf{w}^T\mathbf{x} + b)]}. \tag{3}$$

We fit LR with stochastic gradient updates and L_2 regularization. Its parameter count is linear in the number of inputs, which makes the effect of feature reduction straightforward to quantify. The second classical baseline is a linear SVM trained with a regularized hinge-loss objective. We use the signed margin as its anomaly score. A linear kernel is kept deliberately because its inference cost is fixed by the input dimension; a nonlinear kernel would add a support-vector-dependent cost that is outside the lightweight comparison considered here. The neural baseline is a small fully connected network with 32 ReLU units in the first hidden layer, 16 ReLU units in the second, and one sigmoid output. Hidden-layer widths are unchanged across feature budgets. Reducing the input dimension therefore removes first-layer parameters without changing the depth of the network.

3.6. Validation-Based Decision Threshold

We do not assume that a probability threshold of 0.5 is the most useful operating point. Threshold selection is performed on validation data only. LR and DNN probabilities are searched from 0.05 to 0.95 in steps of 0.01, while the SVM margin is searched from −2.0 to 2.0 in steps of 0.05. For each model, the selected threshold is

$$\tau^* = \arg\max_{\tau} F_1(\tau; \mathcal{D}_{val}). \tag{4}$$

Once τ^* has been chosen, neither the fitted model nor the decision threshold is changed during evaluation on the official test set.

3.7. Detection Metrics

We report accuracy, precision, recall, F1-score, balanced accuracy, MCC, FAR, and ROC-AUC. Attack traffic is treated as the positive class, so

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \tag{5}$$

and

$$F_1 = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{6}$$

The false-alarm rate is

$$\text{FAR} = \frac{FP}{FP + TN}. \tag{7}$$

Balanced accuracy averages recall over the attack and normal classes. MCC uses all four entries of the confusion matrix and is less tied to a single class proportion. ROC-AUC is computed from the continuous scores rather than the selected threshold, which lets us separate ranking behavior from the thresholded decision metrics.

3.8. Edge-Oriented Footprint Measures

No Raspberry Pi or microcontroller is used in this experiment. The resource comparison is therefore limited to model-structure quantities that can be reproduced without choosing a particular hardware platform. Let P be the number of trainable parameters. The approximate FP32 parameter storage is

$$M_{FP32} = 4P \text{ bytes}. \tag{8}$$

A second measure counts weight multiplications required for one forward decision. LR and linear SVM require one multiplication per input feature. For the DNN with input dimension d, the proxy is

$$C_{DNN} = 32d + (32 \times 16) + 16. \tag{9}$$

This value is not a hardware latency measurement. It is used only to compare structural computation across the evaluated models. Actual wall-clock latency depends on implementation language, vectorization, processor architecture, memory hierarchy, quantization, and batch size.

4. Experimental Setup

The experiment uses a deterministic software implementation of the preprocessing, feature ranking, linear learners, and compact DNN. The development split uses seed 42. All models receive the same split within a feature-budget condition. The categorical vocabulary and numerical standardization statistics are learned before validation or test data are evaluated. The LR and linear SVM are trained for 14 epochs with stochastic updates and L_2 coefficient 10^{-4}. The initial learning rate is 0.02 for LR and 0.008 for SVM, followed by an epoch-wise decay of $\eta_e = \eta_0/(1 + 0.1e)$. The DNN is trained for eight epochs with mini-batches of 128 and Adam optimization at learning rate 0.0015. Binary cross-entropy is used for the DNN output. The experiment is deliberately compact; extensive hyperparameter optimization is avoided because the objective is to compare deployment behavior under one transparent protocol rather than to obtain the maximum possible NSL-KDD score for each model. Table 4 records the scenarios used for each model. The DNN has 4,353 trainable parameters with 118 inputs, 1,217 with 20 inputs, and 897 with 10 inputs. Each linear model contains $d + 1$ parameters, including the bias term.

Table 4: *Model configuration used in the controlled comparison.*

Model	Configuration	Training / score used for thresholding
Logistic regression	Linear sigmoid classifier, $L_2 = 10^{-4}$	14 stochastic epochs; validation probability threshold maximizes F1
Linear SVM	Linear hinge-loss classifier, $L_2 = 10^{-4}$	14 stochastic epochs; validation margin threshold maximizes F1
Compact DNN	d–32–16–1; ReLU/ReLU/sigmoid	8 epochs; Adam, batch 128, learning rate 0.0015; validation probability threshold maximizes F1

The feature-budget experiments are performed independently for 118, 20, and 10 input features. The selected 20- and 10-feature sets are derived only from the training partition. Test labels are used exclusively for final metric calculation.

5. Results and Discussion

5.1. Full-Feature Comparison

Table 5 reports results using all 118 encoded inputs. The compact DNN obtained the strongest thresholded performance within this condition. Its accuracy was 0.7667, F1-score was 0.7591, balanced accuracy was 0.7861, and MCC was 0.5785. The DNN also produced the highest ROC-AUC among the full-feature models, 0.8799. LR and linear SVM were close in F1-score, at 0.7419 and 0.7375 respectively. The difference in classification performance is accompanied by a large difference in model structure. LR and linear SVM contain 119 parameters each, while the DNN contains 4,353. Under FP32 storage, this corresponds to approximately 0.465 KB for either linear model and 17.004 KB for the DNN. The structural operation proxy shows the same gap: 118 weight multiplications for a linear decision compared with 4,304 for the DNN. These values do not establish device latency, but they show that the DNN's improvement under the full feature set is purchased with substantially more computation.

Table 5: *Binary anomaly-detection performance using all 118 encoded features.*

Model	Acc.	Prec.	Rec.	F1	Bal. Acc.	MCC	AUC
Logistic regression	0.7533	0.9175	0.6227	0.7419	0.7743	0.5580	0.8471
Linear SVM	0.7486	0.9090	0.6204	0.7375	0.7692	0.5469	0.8262
Compact DNN	**0.7667**	**0.9206**	**0.6458**	**0.7591**	**0.7861**	**0.5785**	**0.8799**

5.2. Effect of Feature Budget

Feature reduction changed the ordering of the models. Figure 2 shows F1-score across the three input budgets. The 20-feature LR produced an F1-score of 0.8054, compared with 0.7419 when the same model used all 118 features. Linear SVM showed a similar pattern, increasing from 0.7375 to 0.7918. The DNN improved more modestly, from 0.7591 to 0.7674.

Among the nine model-budget combinations, the 20-feature LR gave the largest thresholded scores: accuracy 0.7998, F1 0.8054, balanced accuracy 0.8113, and MCC 0.6186. Its ROC-AUC was 0.9137. The 20-feature DNN produced F1 0.7674, but its ROC-AUC reached 0.9230, the highest value observed in the experiment. These two results describe different aspects of the detector. ROC-AUC measures the continuous scores ranking of attack and normal records across possible thresholds, while F1 is only dependent on the selected threshold from the validation partition. The 20 feature DNN had ROC-AUC 0.9230 while LR had the higher, F1 at operating point validated on the validation data. When the input was reduced to 10 features, the error balance for all three was reduced. There were fewer false alarms as compared to the 20-feature condition. However, there was a decrease in attack recall as well. LR reached precision 0.9610 with FAR 0.0314 and recall 0.5853. In the study, the lowest FAR was achieved with the linear SVM 0.0191, together with precision

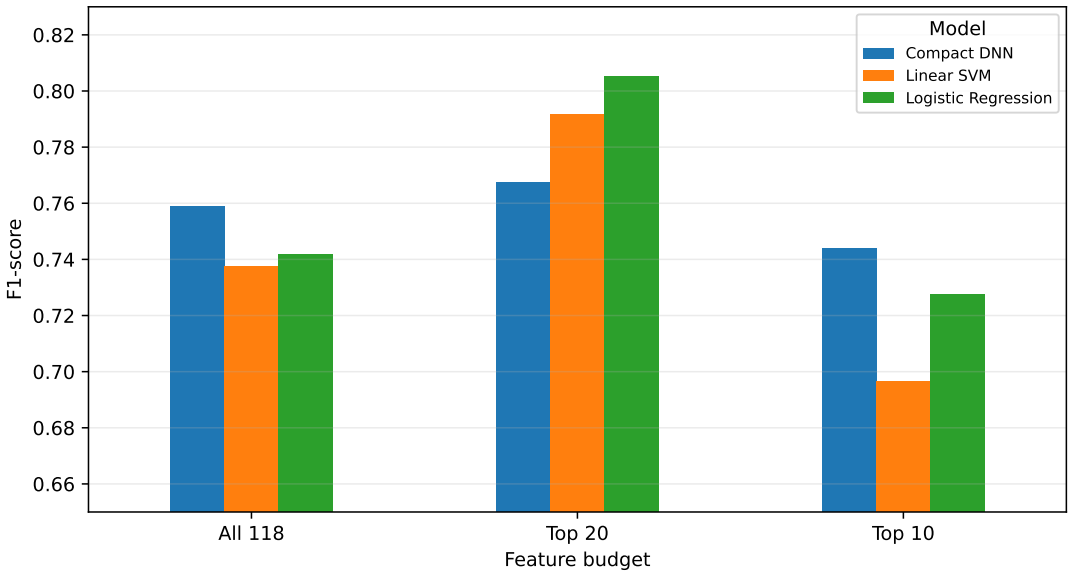


Figure 2: *F1-score for each classifier under the 118-, 20-, and 10-feature budgets.*

0.9741. For the compact DNN, recall was 0.6183 and F1 was 0.7441. The smaller feature set can thus not be said to be in general superior or inferior. It blocks out a lot of false alarms, at the cost of some missed attacks.

Table 6: *Effect of feature budget on detection performance.*

Features	Model	Acc.	Prec.	Rec.	F1	FAR	AUC
118	Logistic regression	0.7533	0.9175	0.6227	0.7419	0.0740	0.8471
118	Linear SVM	0.7486	0.9090	0.6204	0.7375	0.0821	0.8262
118	Compact DNN	0.7667	0.9206	0.6458	0.7591	0.0736	0.8799
20	Logistic regression	**0.7998**	0.9013	**0.7280**	**0.8054**	0.1053	0.9137
20	Linear SVM	0.7885	0.9003	0.7066	0.7918	0.1034	0.9023
20	Compact DNN	0.7711	0.9102	0.6634	0.7674	0.0865	**0.9230**
10	Logistic regression	0.7504	0.9610	0.5853	0.7275	0.0314	0.9080
10	Linear SVM	0.7311	**0.9741**	0.5420	0.6965	**0.0191**	0.9033
10	Compact DNN	0.7579	0.9342	0.6183	0.7441	0.0576	0.8837

5.3. Detection Performance Versus Model Footprint

The relationship between the two structural cost measures used in this study and detection performance is shown in Table 7. LR's input size from 118 features to 20 reduces the estimated FP32 parameter storage from 0.465 KB to 0.082 KB. The number of multiplications decreases from 118 to 20 per record. Although the representation is smaller, F1 improves by 0.7419 to 0.8054, with the improvement of 0.0635.

The difference between the complete-featured DNN is greater. With the same structural proxy, the 20-feature LR needs only about 99.5% less parameters stored and only about 99.5% fewer weight multiplications. It has an official test F1 score that's also 0.0463 higher. These comparisons are for this specific model representation and test set. They are not time to execute on actual edge hardware.

The difference in parameter footprint is shown in figure 3. The range of the linear configurations is considerably less than that of the compact DNN. The 20-feature LR is of particular interest in the current experiment, as it reduces the dimension of its input, which subsequently reduces its "structural footprint", but simultaneously improves its F1-score. This result should not be interpreted as being indicative of the superiority of linear classifiers over neural classifiers in general, but rather as being due to the choice of features and the NSL-KDD evaluation set employed here.

Another aspect of the comparison is seen in the DNN. It has 20 features, and achieves the highest threshold-independent ROC-AUC of 0.9230 among the configurations evaluated. This value is shown in relation to the number of structural multipliers in Figure 4. The neural model needs a lot more computation than either linear alternative, but its scores discriminate between attack and normal records better when thinking of performance across thresholds. This extra computation may or may not be useful in practice, depending on the resources available at the edge node, and on the operating point needed. by the application.

Table 7: *Model footprint and structural computation proxy. FP32 storage includes trainable parameters only.*

Features	Model	Params.	FP32 KB	Ops	F1
118	Logistic regression	119	0.465	118	0.7419
118	Linear SVM	119	0.465	118	0.7375
118	Compact DNN	4,353	17.004	4,304	0.7591
20	Logistic regression	21	0.082	20	0.8054
20	Linear SVM	21	0.082	20	0.7918
20	Compact DNN	1,217	4.754	1,168	0.7674
10	Logistic regression	11	0.043	10	0.7275
10	Linear SVM	11	0.043	10	0.6965
10	Compact DNN	897	3.504	848	0.7441

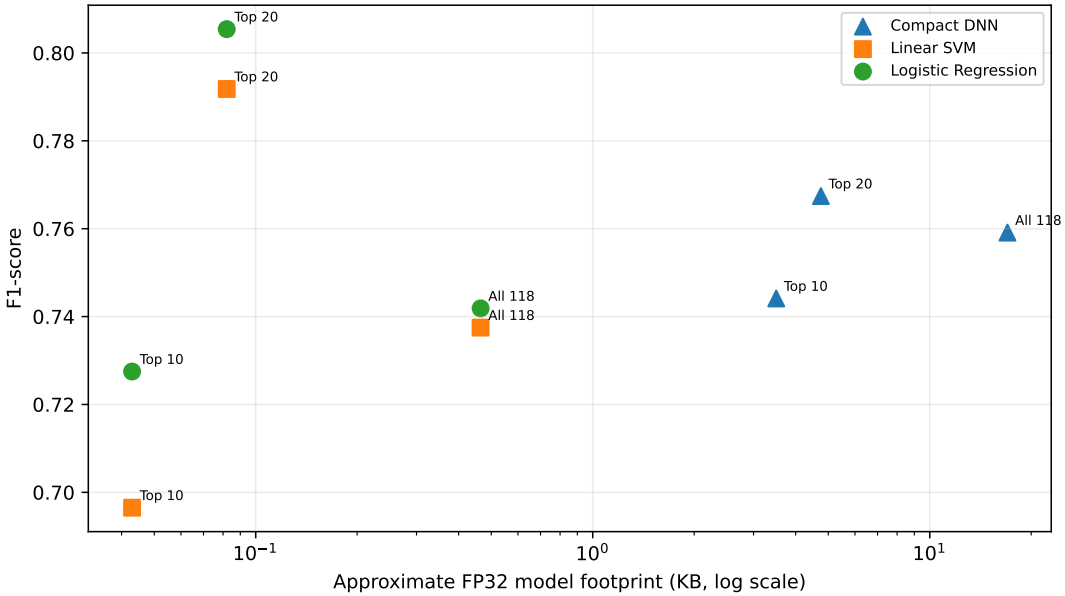


Figure 3: *F1-score versus approximate FP32 parameter footprint. The horizontal axis is logarithmic.*

5.4. False Alarms and Attack Recall

Figure 5 shows the change in false alarms and attack recall by configuration. Both LR and linear SVM have higher FAR when the number of features is increased from 10 to 20. The minimum FAR (0.0191) and minimum recall (0.5420) are achieved by the 10-feature linear SVM. That is, this operating point results in a relatively low number of false alarms and the failure to capture nearly 46% of the attack counts.

The highest attack recall of the experiment is obtained by the 20-feature LR with FAR 0.1053. There are 9,342 correct detections and 3,491 false negatives in the official test set of 12,833 attack occurrences. This same configuration classifies 1,023 records (out of 9,711) as attacks. The 8,288 attacks detected and the 715 false alarms produced by the full-feature DNN are for comparison. These counts provide a more tangible comparison: the 20-feature LR detects more attacks, but also more false alarms. If that exchange is deemed acceptable, it depends on how much a missed attack would cost, and how much burden attaches to extra alerts. intended deployment.

5.5. Why Fewer Features Improved the Linear Models

However, the improvement achieved by the linear models with 20 features shouldn't be interpreted as a proof that removing information in general improves the intrusion detection. Because they are very different in the two training sets, several variables such as connection-state variables and service-rate variables score high under the Fisher criterion. Eliminating weak or redundant dimensions might leave a representation that is easier to be separated by a linear boundary in this benchmark. This is just a guess because the experiment was not designed to isolate this mechanism. The 10-feature results also place a limit on the benefit of reduction. After further reduction of the input set, recall drops for both linear classifiers. The compact DNN has superior F1 and ROC-AUC scores across all 118 inputs than either of the linear baselines. Following Fisher ranking, though, the performance gap alters. The results are consistent with the fact that the chosen 20-feature representation is more beneficial to a linear boundary, but it is not clear why this is the case. A

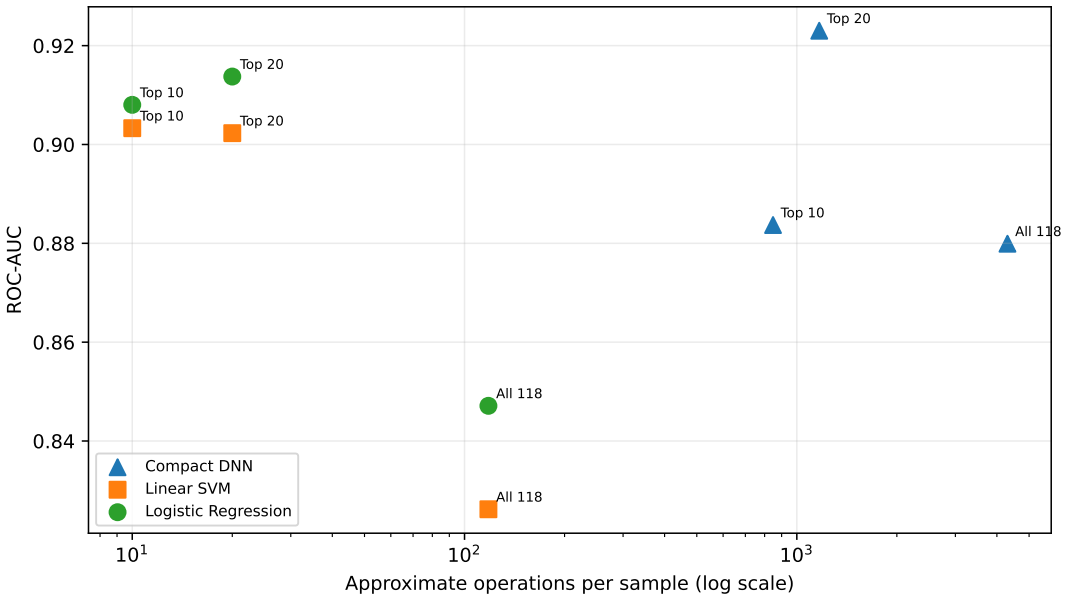


Figure 4: *ROC-AUC versus approximate weight operations per sample. The horizontal axis is logarithmic.*

different ordering may be obtained if a different data set is used, with different combinations of feature interactions or attack classes.

5.6. Practical Implications for Edge IDS Design

The resujlt demonstrates that the number of input features is not just a detail in the pre-processing. Changing the feature budget had a more significant impact on the LR and SVM than did switching between the two linear model families. The interpretation also depends on what the evaluation measure is. The 20-feature DNN achieves the highest ROC-AUC, while the 20-feature LR achieves the highest F1, balanced accuracy and MCC at its chosen threshold. Smaller means smaller, but doesn't excuse attention to its error profile. The 10-feature SVM has the lowest structural cost among the evaluated linear configurations and also produces the lowest FAR, but its attack recall is considerably lower than that of the 20-feature models. For an actual edge deployment, model selection would therefore depend on the available telemetry, resource limits, acceptable false-alert rate, and cost of missed attacks. The structural measures reported here can narrow the set of candidate models, but measurements on the intended hardware are still needed before making an operational deployment decision.

6. Limitations and Threats to Validity

Several constraints limit how far the results can be generalized. NSL-KDD is derived from KDD Cup 99 and does not reproduce current IoT protocols, encrypted traffic behavior, or recent attack patterns. Datasets such as TON_IoT, Edge-IIoTset, and CICIoT2023 provide more recent IoT-oriented scenarios and are more appropriate when the objective is to make claims about contemporary IoT security [12–14]. The conclusions reported here are therefore restricted to the NSL-KDD benchmark. The experiment further only utilizes the standard 20% NSL-KDD development file for model fitting and validation. This selection makes the study compact and reproducible, but the models are not nearly as well trained as models would be with the full NSL-KDD training set. In addition, the task is limited to binary anomaly detection. The experiments don't consider the recognition of individual attack families or attack attribution. The reported resource values refer to model structure not to the actual behaviour of the hardware. Parameter storage and weight operation counts can be easily computed from model definitions but do not incorporate energy consumption, processor utilization, packet processing delay, or framework memory. That is why it is called an "edge-oriented" study, not a physical edge-device benchmark. To perform a hardware evaluation, the frozen models would have to be run on a platform like a Raspberry Pi, Jetson-class device, or other representative edge node, and latency, throughput, resident memory, and, if it is possible to measure, energy would be reported directly. Another drawback is training variability. Reported values are from a single deterministic development split and a carefully limited set of training scenarios. The search for the optimal hyperparameters was not exhaustive, and the multi-seed confidence intervals were not calculated. Further runs of the experiments, alternate splits, and testing on other modern data sets would be required to support any generalization beyond the controlled comparison offered here.

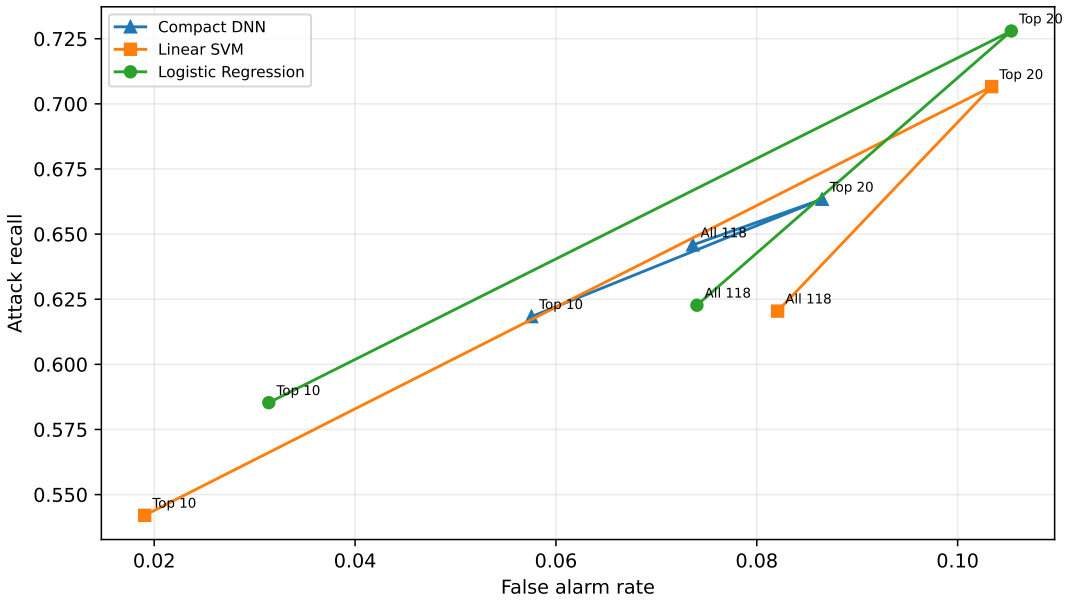


Figure 5: *Trade-off between false-alarm rate and attack recall for the evaluated feature budgets and models.*

7. Conclusion and Future Work

We compared logistic regression, a linear SVM, and a compact DNN for binary network anomaly detection using three feature budgets. Preprocessing and feature ranking were fitted without access to the official test set, and a held-out validation partition was used to choose decision thresholds. Final results were obtained on the official NSL-KDD test file. The comparison reports detection metrics beside parameter storage and a structural computation proxy to keep the edge-side cost visible. No single configuration was strongest on every measure. With all 118 encoded inputs, the compact DNN had the highest F1-score and ROC-AUC within the full-feature condition. After the Fisher ranking reduced the input to 20 features, logistic regression reached F1 0.8054, balanced accuracy 0.8113, and MCC 0.6186 with 21 parameters and about 0.082 KB of FP32 parameter storage. The 20-feature DNN instead produced the highest ROC-AUC, 0.9230. At 10 features, false-alarm rates declined, but attack recall declined as well. For this benchmark, model depth does not remove the need to examine the input representation and operating threshold. A compact linear classifier can perform well when the selected features suit its decision boundary, while the small DNN can offer better score ranking at a higher structural cost.

Future work will repeat the same protocol on Edge-IIoTset, TON_IoT, and CICIoT2023, extend the task to multiclass attack recognition, evaluate repeated training seeds, and run the frozen models on physical edge hardware.

Author Declarations

Author Contributions (CRediT): Ayesha Iftikhar: Conceptualization, Methodology, System Design, Investigation, Visualization, and Writing - Original Draft. Sadia Aslam: Software, Code implementation, Writing - Original Draft, Writing - Review & Editing. All authors have read and approved the submitted version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The study uses the publicly available NSL-KDD dataset. The experiment uses KDDTrain+_20Percent.txt and KDDTest+.txt.

Acknowledgments: The authors would like to thank the Center for Emerging Networks & Technologies (CENT) Lab at the University of Central Punjab (UCP), Lahore, Pakistan, for technical support and constructive feedback.

Conflicts of Interest: The authors declare no conflict of interest.

Ethics Statement: Not applicable.

Declaration of Generative AI: During the preparation of this work, the authors used generative AI tools only for language editing and proofreading, where applicable. The authors reviewed and edited the content and take full responsibility for the published paper.

References

[1] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[2] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, no. 20, p. 4396, 2019.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] M. Burhan, H. Alam, A. Arsalan, R. A. Rehman, M. Anwar, M. Faheem, and M. W. Ashraf, "A comprehensive survey on the cooperation of fog computing paradigm-based iot applications: Layered architecture, real-time security issues, and solutions," *IEEE Access*, vol. 11, pp. 73 303–73 329, 2023.

[5] A. A. Diro and N. K. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018.

[6] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[7] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-baiot: Network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[8] S. U. Jan, S. Ahmed, V. Shakhov, and I. Koo, "Toward a lightweight intrusion detection system for the internet of things," *IEEE Access*, vol. 7, pp. 42 450–42 471, 2019.

[9] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2986–2998, 2016.

[10] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[11] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[12] A. Alsaedi, N. Moustafa, Z. Tari, A. N. Mahmood, and A. Anwar, "TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems," *IEEE Access*, vol. 8, pp. 165 130–165 150, 2020.

[13] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40 281–40 306, 2022.

[14] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment," *Sensors*, vol. 23, no. 13, p. 5941, 2023.

[15] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[16] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.

[17] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with svm for network intrusion detection," *IEEE Access*, vol. 6, pp. 52 843–52 856, 2018.

[18] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for iot security based on learning techniques," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019.

[19] H. Bangui and B. Buhnova, "Lightweight intrusion detection for edge computing networks using deep forest and bio-inspired algorithms," *Computers & Electrical Engineering*, vol. 100, p. 107901, 2022.

[20] P. Mahadevappa, R. K. Murugesan, R. Al-Amri, R. Thabit, A. H. Al-Ghushami, and G. Alkawsi, "A secure edge computing model using machine learning and ids to detect and isolate intruders," *MethodsX*, vol. 12, p. 102597, 2024.

[21] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018, pp. 108–116.