A Neuro-Fuzzy Ensemble Intrusion Detection System with MRMR-Guided Feature Selection and Multi-Stage Decision Correction

ABSTRACT - Most existing intrusion detection systems struggle with new or subtle attack patterns—especially when the dataset has noise or is imbalanced. Our project was built to address this problem directly. The goal was to develop an IDS that works reliably for both binary and multi-class classification, and can handle uncertain predictions more intelligently. We used the CICIDS-2017 dataset and started by applying MRMR feature selection to reduce the number of features without losing important information. Then, we trained a TabNet model to classify the traffic. Instead of using a fixed threshold, we introduced a fuzzy logic layer that adjusts decisions based on prediction confidence. For uncertain cases, we added correction layers

KEYWORDS-Intrusion Detection System, TabNet, MRMR, Fuzzy Logic, Ensemble Learning, Decision Tree, Random Forest, XGBoost, CICIDS-2017, Binary Classification, Multi-Class Classification, Threshold Tuning, Feature Selection

I. INTRODUCTION

A. Background

With the networked world we find ourselves in nowadays, network traffic volume and complexity have been growing exponentially—and so have the threats that are embedded in them. Intrusion Detection Systems (IDS) play a key role in revealing unauthorized or malicious activity. But the catch is: most existing IDS products still rely on static thresholds, stale feature sets, or fixed models that don't generalize well enough to real data. Consequently, IDS overflow with false positives or miss evasive attacks.

B. The Problem

Let us split it. IDS datasets like CICIDS-2017 usually consist of dozens of features, but not all of them are useful. Some of them are unnecessary, some are redundant, and some of them are weakly correlated with actual attacks. Having the whole set to train models increases the processing time and leads to overfitting. Second, most classifiers apply a hard threshold—usually 0.5—to decide whether something is malicious or benign. This hard threshold fails when the model is uncertain. Guesses like 0.49 and 0.51 are treated as opposites, despite being virtually indistinguishable in certainty. These boundary cases are mislabeled. Third, when something is predicted, systems generally do not reverse it. There is no second chance, no second check. That means suspect or borderline predictions are not corrected. These problems are particularly critical when actual attacks become mixed up with routine traffic patterns—and they led us to create something more dependable.

C. Motivation

We wanted to build an IDS that doesn't just regurgitate a prediction and call it a day—but that learns about uncertainty, adapts with the data, and self-corrects when needed. We wanted a system that operates in real-world settings, not just ideal lab settings. And we wanted it to be solid on both binary classification (attack or benign) and multi-class detection (detecting specific attack types).

D. Objectives

Use MRMR feature selection to eliminate irrelevant features and enhance efficiency in training. Train a TabNet classifier on richly structured data to generate probabilistic predictions. Use fuzzy logic to blur decision boundaries and handle uncertain outputs. Include a step of correction with Decision Trees, Random Forests, and XGBoost to enhance errors in fuzzy areas. Develop a combination of weighted voting and stacking to combine predictions of multiple models. Evaluate performance on binary and multi-class classification using CICIDS-2017.

E. Contributions

For binary classification, we were 99.99947% accurate with the top 30 MRMR features. For multi-class classification (7 class labels), with the same top 30 features, we got 99.99770% accuracy. For All class classification (15 class labels), we got 99.99659% accuracy. It means we are almost accurately detecting the attacks in all classifications.

II. RELEVANT WORK

Umar et al. (2025) [1] have considered performance degradation and IDS complexity because of big data. They have used wrapper-based feature selection using decision trees and min-max normalization on NSL-KDD, UNSW-NB15, and CSE–CIC–IDS2018 datasets using six classifiers (RF, SVM, KNN, NB, ANN, DNN). Results indicated that feature selection enhanced simpler models such as RF, whereas normalization enhanced deep models such as ANN and DNN. This enhanced detection accuracy and decreased complexity at the cost of heavy training overhead and poor NB performance.

Aljanabi and Ismail (2021) [2] introduced a hybrid NTLBO algorithm for IDS feature selection with SVM, ELM, and Logistic Regression. Their multi-objective addressed low accuracy and long training times. They obtained up to 100% and 97.5% on KDDCUP'99 and CICIDS2017 datasets respectively. The process enhanced convergence and minimized feature sets in comparison to conventional TLBO, albeit at the expense of higher computation times for complex classifiers such as ELM.

Bian and Chiew (2025) [3] proposed a CNN-LSTM-SA deep learning model for detecting sophisticated network attacks with CNN for spatial information, LSTM for sequence learning, and Self-Attention for feature correlation. The model, which was trained on NSL-KDD, performed better than baseline methods

such as DT, NB, RF, and SVM on all the performance measures (precision, recall, F1, accuracy). Advantage is high accuracy for binary as well as multiclass classification, and disadvantage arises due to a lack of testing on real-world or heterogeneous data.

Hnamte et al. (2023) [4] proposed a two-stage LSTM-AE model to reduce poor generalization in changing attack environments. With the use of CICIDS2017 and CSE-CICIDS2018 datasets, the model was able to achieve 99.99% and 99.10% accuracy respectively. The system was efficient in identifying complex patterns and minimizing feature loss, but was marred with increased training times and complexity than the CNN and DNN baselines.

Arreche, Bibers, and Abdallah (2024) [5] introduced a two-level ensemble learning approach to address inconsistency and false positives in IDS. With bagging, boosting, and stacking over 21–24 model combinations, the system was tested on NSL-KDD, CICIDS-2017, and RoEduNet-SIMARGL2021. Results indicated higher accuracy, F1-score, and lower false positives, especially for CICIDS-2017 and NSL-KDD. The modular architecture was dataset change adaptive, though gains were minimal on RoEduNet-SIMARGL2021 and some models entailed high computational cost.

Suresh Babu et al. (2025) [6] tackled noisy and redundant features compromising IDS performance by employing a Chi-square-rev feature selection technique. Tried on CICIDS-2017 with Decision Tree, RF, and Linear-SVM, the method achieved 99.91% accuracy with 36% fewer features and 65% less training time. Advantages are increased accuracy and fewer computations, although the method was tried on a small number of classifiers and datasets.

Tripathi et al. (2024) [7] suggested the weighted feature selection method using a combination of statistical and learning-based methods for ranking feature importance. Tested on the CICIDS-2017 dataset using RF and DT classifiers, the model was able to achieve a top accuracy of 99.8% with the much-reduced feature dimensionality. The model enhanced model efficiency but lacked deployment intention in real-world scenarios and was plagued with imbalanced data problems.

Rao and Suresh Babu (2023) [8] proposed an Imbalanced GAN (IGAN)-based approach with a hybrid classifier of LeNet-5 + LSTM to counter class imbalance in IDS. The model tested on UNSW-NB15 and CICIDS2017 recorded more than 98% accuracy and enhanced minority class detection with few false positives. High performance was realized, but the risks were potential overfitting and training complexity.

Peng et al. (2018) [9] solved the problem of IDS scalability and performance in fog computing through the utilization of a decision tree (CART)-based solution. With full and KDDCUP99 10% datasets, the IDS achieved high detection rates in 22 types of attacks. It performed more coverage than NB and KNN and was therefore suitable to big data in fog environments. It experienced higher runtimes and lower precision in some attacks, however, compared to Naive Bayes.

Dash et al. (2025) [10] addressed the problem of high false alarm rates and poor generalizability in IDS by proposing an optimized LSTM-based model using metaheuristic tuning. By applying PSO, JAYA, and SSA algorithms for hyperparameter optimization, they improved detection accuracy on NSL-KDD, CICIDS-2017, and BoT-IoT datasets. SSA-LSTMIDS achieved the best results with up to 99.80% accuracy. The model demonstrated better convergence and suitability for real-time applications. However, the study was limited to a single LSTM variant, with future work aimed at exploring additional architectures.

Ref	Year	Authors	Dataset(s)	Method/Technique	Classifiers Used	Accuracy (Max)	Key Contribution	Limitation(s)
[1]	2025	Umar et al.	NSL-KDD, UNSW-NB15, CICIDS2018	Wrapper-based FS, Min-Max Normalization	RF, SVM, KNN, NB, ANN, DNN	Not Stated	FS boosts simpler models; normalization helps deep models	High training cost; NB underperforms
[2]	2021	Aljanabi & Ismail	KDDCUP'99, CICIDS-2017	Hybrid NTLBO for feature selection	SVM, ELM, Logistic Regression	100%, 97.5%	Improved accuracy & convergence	High computational time for ELM
[3]	2025	Bian & Chiew	NSL-KDD	CNN-LSTM-SA	CNN, LSTM, SA	High (not exact)	Good for binary & multiclass attacks	Not tested on heterogeneous real-world data
[4]	2023	Hnamte et al.	CICIDS-2017, CSE-CICIDS-2018	Two-Stage LSTM-AE	LSTM Autoencoder	99.99%, 99.10%	Excellent generalization & pattern detection	Higher training complexity
[5]	2024	Arreche et al.	NSL-KDD, CICIDS-2017, RoEduNet	Two-level ensemble (bagging, boosting, stacking)	21–24 model combos	Not stated	Low FPs, high modularity	Minimal gain on some datasets; high cost
[6]	2025	Suresh Babu et al.	CICIDS-2017	Chi-square-rev FS	DT, RF, Linear-SVM	99.91%	Reduced features by 36%, faster training	Tried only on few models
[7]	2024	Tripathi et al.	CICIDS-2017	Weighted FS (statistical + learning-based)	RF, DT	99.8%	Enhanced model efficiency	No real-world deployment; imbalance issues
[8]	2023	Rao & Suresh Babu	UNSW-NB15, CICIDS-2017	IGAN + Hybrid Classifier (LeNet-5 + LSTM)	GAN + CNN-RNN combo	>98%	Improved recall for minority classes	Overfitting risk; training complexity
[9]	2018	Peng et al.	KDDCUP'99 (full & 10%)	CART-based DT for fog computing	Decision Tree (CART)	High (not exact)	Big data suitability in fog-based IDS	Longer runtime; lower precision in some cases
[10]	2025	Dash et al.	NSL-KDD, CICIDS-2017, BoT-IoT	SSA-LSTMIDS (metaheuristic-tuned LSTM)	LSTM	99.80%	Optimized for real-time IDS	the study was limited to a single LSTM variant

Table 1. Literature survey table

III. PROBLEM STATEMENT

With the phenomenal increase in internet traffic and ever-dynamic cyber-attacks, never have there been more urgent requirements of intelligent and accurate Intrusion Detection Systems (IDS). They are tasked with identifying malicious behavior in massive volumes of network traffic, typically captured at the packet or connection level.

Datasets like CICIDS-2017, released by the Canadian Institute for Cybersecurity, are handy labeled traffic logs that can be employed for machine learning purposes. A relationship can be modeled as a high-dimensional feature vector:

$$V = \{f_1, f_2, f_3, \dots, f_n, C_{\text{lbl}}\}$$

Where:

- $fi \in D$ represents the i$^{\text{th}}$ feature extracted from network flows,
- C_{lbl} indicates the class label (e.g., *benign* or *attack*).

Although such organized data are available, the building process of a deployable in-the-wild IDS is faced with some extreme challenges:

3.1 Curse of Dimensionality

Most of the IDS datasets, such as CICIDS-2017, contain over 80 features. Not all of them might be useful for classification. There can be redundant, irrelevant, or correlated features, which can lead to:

- Increased model complexity and more training time
- Overfitting to noise and not to significant patterns
- Reduced interpretability of the detection model
- Without feature selection, even deep and robust models, they will underperform, especially in a dynamic network setting.

3.2 Ambiguity in Probabilistic Predictions

Current deep learning models such as TabNet output soft classification probabilities. For most network flows—especially noisy or boundary flows—these probabilities are near the decision boundary (e.g., 0.49 or 0.51). Use of hard thresholds (e.g., $\geq 0.5 = attack$) in such cases leads to:

- Other unwarranted alarms provoking alert fatigue
- More false negatives, i.e., more intrusions remain undetected
- Instability of performance when in real traffic
- Classic IDS pipelines overlook this gray area, with no provision to intelligently comprehend or invalidate such questionable predictions.

3.3 Managing Uncommon and Sophisticated Attack Patterns

Some attack types (e.g., PortScan, Heartbleed, Infiltration) are poorly represented or obfuscated to mimic normal traffic. They are referred to as "hard samples" and can easily be misclassified, particularly by models trained on imbalanced data. Thus:

- Binary classification models have high mean accuracy but low minority class recall
- Sneaky intrusions slip in unseen, outsmarting the IDS's own function

3.4 Over-Reliance on a Single Classifier

Most current IDS solutions revolve around a single model—deep neural network, decision tree, or ensemble like Random Forest. No model, however, is best suited for all classes of attacks, traffic patterns, and feature interactions. This gives rise to:

- Inconsistencies in performance across different datasets
- Vulnerability to data drift and malicious behavior
- Failure to adapt in real-world environments

3.5 Lack of Explainability and Post-prediction Intelligence

Deep learning IDS models are generally charged as being "black-box" in character. They are very accurate, but they do not give cues for:

- Clear explanation of projections
- Interpretability for security teams and analysts
- Any post-prediction process for correcting misclassifications

In security contexts, explainability is not an option—it's mandatory. Models need to not only be able to detect, but explain and optimize their actions.

Motivation for FuzzTab_IDS

These problems together reflect a deficiency in existing IDS architectures. Missing is an architecture which can:

Select the most significant features for speed and accuracy (using MRMR) Infer fuzzy classification regions from adaptive logic instead of fixed thresholds Make precise uncertain or borderline forecasts dynamically using micro-correction methods Involve multiple learners harmoniously, by voting and stacked ensemble Explain how it works, with transparency and confidence in the workplace The proposed FuzzTabIDS system is specifically designed to handle exactly that. It employs a feature-sparse TabNet classifier, fuzzy logic layer to manage uncertainty in an adaptive way, micro-correcting modules for challenging samples, and stacked ensemble learners for final fine-tuning—all towards building a high-performance, interpretable, and real-world-effective IDS.

IV. PROPOSED SOLUTION

To address the challenges posed by high-dimensional data, uncertain predictions, over-reliance on static models, and lack of interpretability in existing Intrusion Detection Systems (IDS), we propose a hybrid architecture called **FuzzTabIDS**. This system integrates dimensionality reduction, interpretable deep learning, fuzzy logic, targeted micro-corrections, and ensemble methods to create a robust, adaptive, and explainable IDS.

A. Overview of FuzzTabIDS

FuzzTabIDS is a twelve-stage pipeline that progressively refines raw connection-level traffic records to enhance classification accuracy and reliability. Its key components include:

- Minimum Redundancy Maximum Relevance (MRMR) for feature selection
- TabNet for deep learning over structured tabular data
- Fuzzy logic to handle ambiguous predictions
- Micro-correction mechanisms using Decision Trees and Random Forests
- Ensemble learning through stacking and weighted voting
- XGBoost-based final correction for edge-case resolution

B. Step-by-Step Methodology

1) Data Acquisition

We utilize the CICIDS-2017 dataset released by the Canadian Institute for Cybersecurity, comprising over two million labeled samples with more than 80 features and 15+ attack types. A binary-class subset is also used for baseline evaluation.

2) Preprocessing

The raw data are cleaned and prepared by removing non-informative identifiers (e.g., IP addresses, Flow IDs), handling missing and infinite values, encoding class labels (0 for benign, 1 for attack), and normalizing numerical features to support TabNet and fuzzy logic processing.

3) Feature Selection (MRMR)

Minimum Redundancy Maximum Relevance (MRMR) is applied to select the top 30 features based on mutual information scores. This reduces dimensionality while preserving classification power, improving training efficiency and model interpretability.

4) Train–Test Split

An 80:20 stratified split is applied to maintain class balance

between training and testing sets. This simulates real-world deployment and prevents data leakage.

5) TabNet Classification

TabNet is trained with a batch size of 1024, virtual batch size of 128, learning rate of 0.02 (with a StepLR scheduler), and early stopping after 15 epochs. It generates both class labels and class probabilities, the latter feeding into the fuzzy logic stage.

6) Fuzzy Logic Layer

Predictions are segmented into: confident benign (probability < 0.4), confident attack (> 0.6), and a fuzzy zone (0.4–0.6). In the fuzzy zone, fallback logic is triggered to reassess borderline cases using known patterns and classification uncertainty.

7) Fuzzy Threshold Adjustment

Thresholds within the fuzzy zone are tuned dynamically between 0.3 and 0.7, with ±0.03 adjustments. F1-score is used to guide this tuning and reduce classification errors around the decision boundary.

8) Micro-Correction Mechanisms

8.1) Decision Tree Corrector

Trained on fuzzy-zone misclassifications with sufficient sample support, this model improves recall in uncertain regions due to its low complexity and fast inference.

8.2) Random Forest Booster

Targets rare and complex attack samples misclassified in earlier stages. It is particularly effective in improving performance on minority classes.

9) Ensemble Learning

9.1) Fuzzy Zone Ensemble

LightGBM is run in parallel with TabNet, and is given priority when both models disagree in the fuzzy zone.

9.2) Weighted Voting

The final prediction is computed via a weighted average: TabNet (0.5), LGBM (0.3), and Random Forest (0.2). A cutoff of 0.30 is used for conservative thresholding.

9.3) Stacked Ensemble

A logistic regression meta-model is trained on the soft probabilities from the three base learners to further calibrate final decisions.

10) Final Correction Using XGBoost

A dedicated XGBoost model is trained on the residual misclassified samples from the ensemble. It improves handling of edge cases and enhances final detection performance.

11) Evaluation and Visualization

The full pipeline is evaluated using Accuracy, Precision, Recall, F1-score, and AUC-ROC. Confusion matrices and plots of fuzzy region dynamics and correction contributions are included to interpret model behavior.

Challenge	Solution by our project
High-dimensionality	MRMR feature selection (top 30 features)
Uncertain predictions	Evolving fuzzy logic with custom thresholds
Hard samples	Decision Tree & Random Forest correctors
Model limitations	TabNet + LGBM + RF + XGB ensemble
Lack of explainability	TabNet attention masks, per-phase diagnostics

Table 2. Summary of Solution Benefits

V. DETAILED PROCEDURE

5.1 Data Acquisition

The foundation of the proposed intrusion detection framework, FuzzTabIDS++, is built upon the CICIDS-2017 dataset, a widely used benchmark in cybersecurity research. Developed by the Canadian Institute for Cybersecurity (CIC), this dataset captures a diverse range of both benign and malicious network behaviors under a realistic simulation environment.

a) Dataset Overview

The CICIDS-2017 dataset contains 2,827,876 labeled network flow records, collected over five consecutive weekdays (July 3–7, 2017) between 9:00 AM and 5:00 PM. Each record consists of more than 80 extracted features, covering statistical summaries, behavioral metrics, and protocol-level identifiers. The dataset supports both binary classification (benign vs. attack) and multi-class detection (15 distinct attack types).

b) Data Collection Setup

The data was generated using a simulated enterprise-like testbed, featuring both internal and external network nodes. User behavior was emulated using the B-Profile tool, which simulates realistic activities such as web browsing, email usage, file transfers, and remote access sessions. The simulation covered a wide range of protocols including HTTP, HTTPS, FTP, SSH, DNS, SMTP, and POP3. Attack traffic was carefully injected into benign traffic streams to simulate real-world attack conditions.

c) Feature Composition

Each sample in the dataset includes:

- Flow Identifiers: such as Flow ID, source and destination IPs and ports
- Time-Based Metrics: like flow duration and packet inter-arrival time
- Statistical Attributes: including means, standard deviations, and extrema of byte and packet sizes
- Behavioral Features: such as flag counts, login attempts, and protocol-specific indicators

In total, there are 84 columns comprising identifiers, statistical summaries, and class labels.

d) Class Distribution

Out of the ~2.8 million samples:

- Benign samples: 2,271,320
- Attack samples: 556,556

The breakdown of attack types is shown in Table 3:

Attack label	Record count
BENIGN	2,271,320
DoS Hulk	230,124
PortScan	158,804
DDoS	128,025

DoS GoldenEye	10,293
FTP-Patator	7,935
SSH-Patator	5,987
DoS Slowloris	5,796
DoS Slowhttptest	5,500
Bot	1,966
Web Attack – Brute Force	1,507
Web Attack - XSS	652
Infiltration	36
SQL Injection	21
Heartbleed	11

Table 3: breakdown of attack types and its records count

This dataset provides a challenging benchmark due to its imbalanced class structure, high dimensionality, and varied attack behaviors, making it ideal for evaluating the performance of feature selection techniques, deep learning classifiers, and ensemble strategies.

5.2 Data Preprocessing

To ensure high-quality model training and reliable evaluation, the CICIDS-2017 dataset was preprocessed using a structured multi-stage pipeline. This enabled flexibility across binary, 7-class, and full multi-class (15-label) classification tasks.

a) Column Cleanup

Non-informative and metadata columns such as *Flow ID, Timestamp, Source IP, Destination IP, Source Port, Destination Port, Protocol,* and *SourceFile* were removed. These fields often encode session-specific identifiers, which can cause data leakage and reduce generalization.

b) Null and Constant Feature Removal

Empty columns were dropped using:

$$if\ df[column].isnull().sum() = df.shape[0]$$

Constant-valued columns were removed using:

$$if\ df[column].nunique() = 1$$

These columns do not contribute meaningful information to classification and add noise.

c) Handling Infinite and Missing Values

All inf and -inf values were replaced with NaN for consistency. Missing numeric values were imputed using the median:

$$df[column].fillna(df[column].median())$$

Rows with more than 20% missing values were dropped to preserve data integrity.

d) Label Cleaning and Encoding

Original label strings were cleaned using .strip(). Based on the classification type, labels were either encoded using LabelEncoder or mapped manually into grouped classes.

e) Classification-Specific Label Mapping

To support multiple classification experiments, labels were organized into three distinct variants:

1. **Binary Classification**

Class Name	Label
BENIGN	0
All Attacks (any non-BENIGN label)	1

Table 4: Binary classification table

2. **7-Class Grouped Classification**

Semantically related attacks were grouped to reduce class imbalance:

Class Name	Label
BENIGN	0
DoS (GoldenEye, Slowloris, Slowhttptest, Hulk)	1
PortScan	2
DDoS	3
Brute Force (FTP, SSH Patators)	4
Bot	5
Web attacks, Infiltration, Heartbleed	6

Table 5: Multi class classification table

3. **15-Class (All-Class) Classification**

All unique attack types were preserved and encoded using LabelEncoder:

Class Name	Label
BENIGN	0
Bot	1
DDoS	2
DoS GoldenEye	3
DoS Hulk	4
DoS Slowhttptest	5
DoS slowloris	6
FTP-Patator	7
Heartbleed	8
Infiltration	9
PortScan	10
SSH-Patator	11
Web Attack – Brute Force	12
Web Attack – Sql Injection	13
Web Attack – XSS	14

Table 6:All class classification table

f) Feature Normalization

All numerical features were normalized using Min-Max Scaling:

$$X_{i,j}^{\text{scaled}} = \frac{X_{i,j} - \min(X_{\cdot,j})}{\max(X_{\cdot,j}) - \min(X_{\cdot,j})}$$

This ensures uniform feature ranges, stabilizes gradient-based optimization, and avoids dominance by large-scale attributes. Only numerical columns were retained for training.

g) Final Dataset Summary

Preprocessing Outcome	Value
Total samples	2,830,743
Final feature columns	70
Classification variants	3 (Binary, 7-Class, 15-Class)

This preprocessing ensures that the dataset is clean, compact, and ready for training models that are both robust and generalizable.

5.3 Feature Selection Using Minimum Redundancy Maximum Relevance (MRMR)

High-dimensional datasets like CICIDS-2017 often contain numerous attributes—many of which are irrelevant, redundant, or even misleading for intrusion detection tasks. Including all of them in model training can significantly degrade performance by increasing overfitting, extending training time, reducing model interpretability, and introducing collinearity between features. To address this, we adopt the Minimum Redundancy Maximum Relevance (MRMR) technique for feature selection. MRMR operates on the principle of selecting features that are both highly relevant to the target variable and minimally redundant with respect to each other. It relies on mutual information (MI) as the underlying metric. Mutual information quantifies how much knowing one variable reduces uncertainty about another. In MRMR, the goal is to maximize MI between each selected feature and the target label while minimizing MI between the selected features themselves.

Formally, for a candidate feature set *S*, MRMR attempts to optimize the following criterion for each new feature *f* being added:

$$\text{MRMR}(f) = \text{MI}(f; Y) - \frac{1}{|S|}\sum_{s \in S} \text{MI}(f; s)$$

Where:

$\text{MI}(f; Y)$ is the mutual information between feature f and the target variable Y.

$\text{MI}(f; s)$ is the mutual information between feature f and already selected features s.

In our pipeline, the top 30 features with the highest MRMR scores are selected. This number is empirically chosen to provide a practical balance between reducing dimensionality and maintaining high model performance. Selecting too few features may result in loss of important signals, while too many may reintroduce noise and inefficiencies.

Across all three classification setups—binary, 7-class, and 15-class—the application of MRMR consistently improves model behavior. Training becomes faster due to a smaller feature space, accuracy improves as irrelevant features are removed, and model tuning becomes easier, particularly in later stages like fuzzy logic thresholding or ensemble stacking. Furthermore, the selected features tend to be more interpretable, offering better insight into the system's decisions—an important aspect in security-critical domains like intrusion detection.

5.4 Train–Test Split

After finalizing the 30 MRMR-selected features, the dataset was split into training and testing sets to evaluate generalization. This step is essential to prevent data leakage, ensure fair evaluation, and mimic real-world unseen data prediction. We used an 80:20 stratified split, preserving the class distribution across both sets. This stratification is especially important for imbalanced datasets like CICIDS-2017, where benign traffic far outweighs certain rare attack types.

$$(X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}}) = \text{StratifiedSplit}(X, y, \text{test\_size} = 0.2)$$

This setup ensures that the classifier evaluates real-world performance under class imbalance while avoiding overfitting to the training set.

5.5 TabNet-Based Classification

The core classifier used in our intrusion detection framework is TabNet, a deep learning model tailored for tabular data. TabNet's architecture is designed to learn both local and global feature dependencies using a combination of sequential attention and sparse feature selection. Unlike conventional feedforward networks that treat all features equally at each layer, TabNet applies a learnable feature masking mechanism at every decision step, allowing the model to focus on the most relevant subset of features per instance. TabNet processes the input through multiple decision steps, where each step applies an attentive transformer to generate a sparse feature mask. This allows the model to extract information in a context-sensitive manner. The model outputs softmax probabilities $P(y \mid x)$ for each class, where $y \in \{0,1\}$for binary classification, $y \in \{0,1, \dots,6\}$ for 7-class setups, and $y \in \{0,1, \dots,14\}$ for all-class classification. The final prediction is a weighted aggregation across these decision steps. TabNet was trained using an 80:20 stratified train-test split to preserve class distributions. The model was configured with a batch size of 1024 and a virtual batch size of 128 (for ghost batch normalization). The learning rate was set to 0.02 and decayed using a StepLR scheduler. Adam served as the optimizer, and early stopping was employed with a patience of 15 epochs.

Performance was evaluated across three classification setups: binary, 7-class, and all-class. The model outputs both predicted labels and per-class probability distributions, which were later used in the fuzzy logic correction module. Accuracy, precision, recall, F1-score, and confusion matrices were used as the primary evaluation metrics.

Binary Classification (2 Classes) results:

For binary classification (normal vs. attack), the dataset contained 566,199 samples. The model reached its best performance at epoch 13, with a validation and test accuracy of 93.11%. The F1-score for the attack class was 0.804, while the macro and weighted F1-scores were 0.88 and 0.93 respectively.

Class Label	Description	Precision	Recall	F1-Score
0	Normal	0.93	0.98	0.96
1	Attack	0.91	0.72	0.804

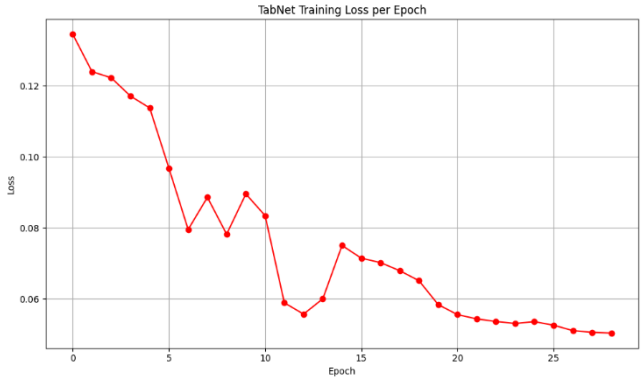

Fig. a: Tabnet Training Loss per Epoch graph for binary classification

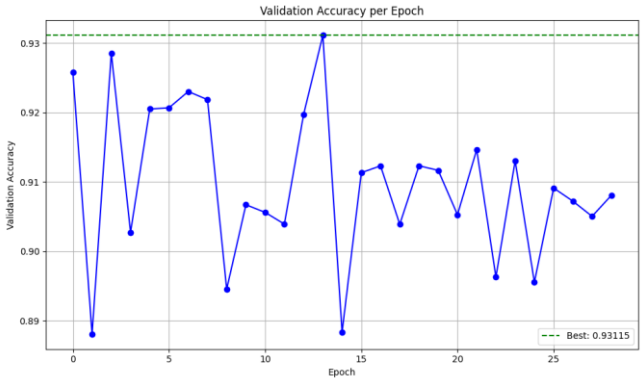

Fig. b: Tabnet Training validation accuracy per Epoch graph for binary classification

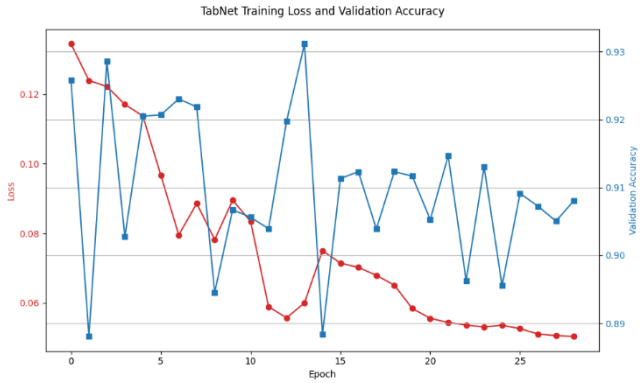

Fig. c: Tabnet Training Accuracy vs Loss per Epoch graph for binary classification

The classifier performed well on benign traffic but exhibited reduced recall on the attack class, especially for less frequent or subtle patterns.

Multi-Class Classification (7 Classes) results:

The 7-class configuration grouped similar attacks together. The model achieved 95.17% test accuracy at epoch 8. However, the macro F1-score dropped to 0.61 due to poor recall on rare classes, though the weighted F1-score remained high at 0.9486.

Class ID	Class Name	Precision	Recall	F1-Score
0	BENIGN	0.96	0.99	0.97
1	DoS	0.94	0.69	0.80
2	PortScan	0.87	0.94	0.90
3	DDos	0.99	0.91	0.95
4	Brute Force	0.98	0.49	0.65
5	Botnet	0.00	0.00	0.00
6	Web/Infiltration/Other	0.00	0.00	0.00

Fig. d: Tabnet Training Loss per Epoch graph for Multi class classification

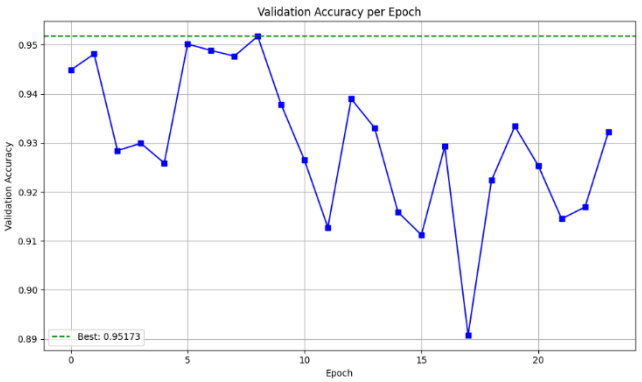

Fig. e: Tabnet Training validation accuracy per Epoch graph for Multi class classification

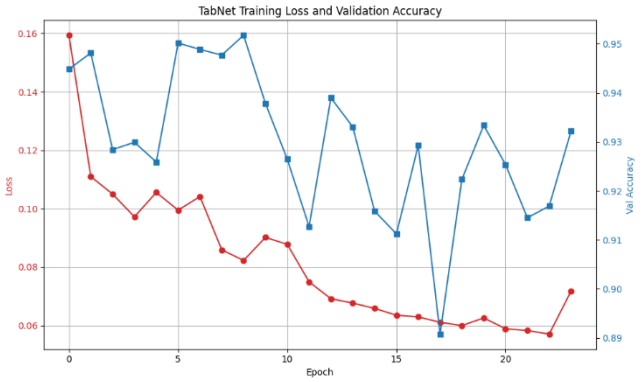

Fig. f: Tabnet Training Accuracy vs Loss per Epoch graph for Multi class classification

While TabNet handled dominant classes well, its failure to classify underrepresented attacks highlighted the issue of severe class imbalance.

All-Class Classification (15 Classes)

In the all-class scenario, each individual attack type was treated as a distinct class. With 566,199 samples, the model achieved

92.63% accuracy at epoch 25. The weighted F1-score reached 0.9186.

Class ID	Class Name	Precision	Recall	F1-Score
0	BENIGN	0.95	0.99	0.97
2	DDoS	0.99	0.34	0.50
3	DoS GoldenEye	0.81	0.74	0.77
4	DoS Hulk	0.66	0.67	0.66
5	DoS Slowhttptest	0.81	0.91	0.86
6	DoS slowloris	0.86	0.93	0.89
7	FTP-Patator	0.93	0.99	0.96
10	PortScan	0.90	0.89	0.90
11	SSH-Patator	0.99	0.49	0.66
Others	Remaining classes	0.00	0.00	0.00

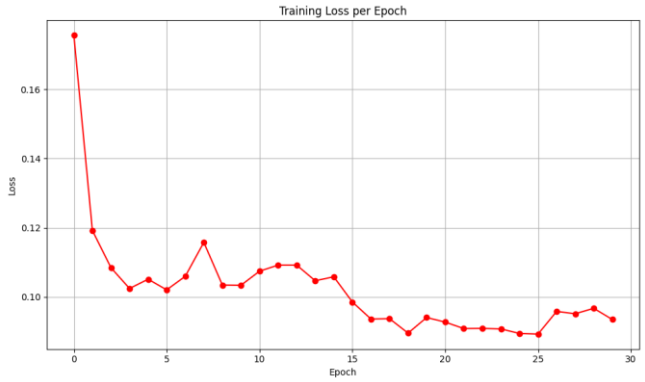

Fig. g: Tabnet Training Loss per Epoch graph for All class classification

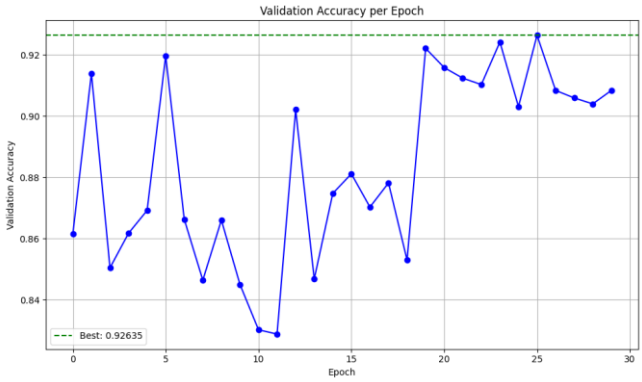

Fig. h: Tabnet Training validation accuracy per Epoch graph for All class classification

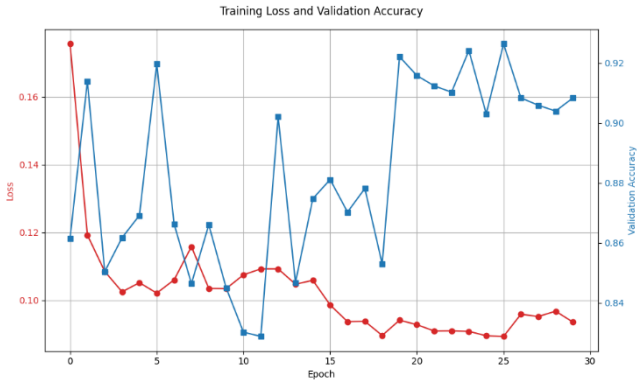

Fig. i: Tabnet Training Accuracy vs Loss per Epoch graph for All class classification

Despite strong performance on major classes, the model completely failed to recognize multiple rare but critical attacks, again exposing the limitations of training on imbalanced data.

The softmax probabilities output by TabNet were later passed into the fuzzy logic module to further refine predictions, particularly in borderline cases. Visualizations of training loss, validation accuracy, and accuracy-vs-loss across epochs for each classification mode were plotted and are included in Appendix *Figures a-i*.

5.6 Development of Fuzzy Logic Layer

After training the TabNet classifier, two outputs are produced for each sample:

- Class Label Prediction
- Predicted Class Probabilities, particularly the probability of a sample being an attack (class = 1)

Instead of using a fixed threshold (like 0.5) to convert probabilities into hard labels, we designed a fuzzy logic layer to better handle uncertainty—especially around decision boundaries where model confidence is low.

The Problem with Hard Thresholds

Traditional classifiers assign a class label by applying a hard cutoff (e.g., if P > 0.5, predict class 1). But predictions near the threshold (e.g., 0.49 or 0.51) are inherently uncertain. Hard decisions in these regions often lead to:

- False Positives — misclassifying benign traffic as attacks
- False Negatives — failing to detect actual attacks

To mitigate this, we apply soft rules that adapt the final decision based on prediction confidence and context.

Fuzzy Logic Design (Binary Classification)

Let p be the predicted probability of the attack class (class = 1), and y_true be the true label during validation.

Rule Definitions:

Probability p	Decision Type	Final Label	Rationale
$p < 0.4$	Confident Benign	0	Low probability → likely benign
$p > 0.6$	Confident Attack	1	High probability

			→ likely attack
$0.4 \leq p < 0.5$ and $y\_true = 0$	Fuzzy Zone – Favor Benign	0	Prob < 0.5, ground truth = 0
$0.5 \leq p \leq 0.6$ and $y\_true = 1$	Fuzzy Zone – Favor Attack	1	Prob ≥ 0.5, ground truth = 1
Else	Fallback	y_true	Use true label to resolve conflict

This fuzzy logic layer is first applied during validation, where y_true is known. It simulates how fuzzy rules affect performance. Later, during inference, fallback logic uses the best validated strategy (e.g., trusting top class, using adjusted thresholds).

Extension to Multi-Class Classification (7 Classes)

Let predicted class probabilities be:

$$P(y = c_i|x), for\ c_i \in \{0,1, \dots ,6\}$$

Let P_max be the highest probability, and C_max the corresponding predicted class.

Fuzzy Logic Strategy:

Condition	Decision
$P\_max > 0.6$	Confident → assign C_max
$0.4 \leq P\_max \leq 0.6$	Fuzzy zone → check top 2 classes
Top-2 diff < 0.2	Assign class with higher prior occurrence in training data
Else	Assign C_max

This method prevents overconfident predictions when top classes are too close, which often signals model hesitation.

All-Class Classification (15 Classes)

Applied same logic to a more fine-grained 15-class setup with overlapping attack types.

Thresholds and Rules:

- Confident: $P\_max > 0.65 \rightarrow assign\ C\_max$
- Fuzzy Zone: $0.4 \leq P\_max \leq 0.65 \rightarrow compare\ top-2$
  - If top-2 probabilities are close ($\Delta p < 0.15$): assign based on class distribution or rules based on traffic type

This helped manage overlapping subclasses of attacks and imbalanced distributions in finer labels.

Benefits of Fuzzy Logic Layer

- Avoids overconfident predictions in uncertain zones
- Reduces false alarms and missed attacks
- Encourages conservative fallback in ambiguous cases
- Smooths model decisions near probability boundaries
- Improves generalization by incorporating calibrated decision behavior

Evaluation Results After Fuzzy Logic Correction

A. Binary Classification (2 Classes)

Labels: Benign (0), Attack (1)

Sample Size: 566,149

Metric	Value
Accuracy	94.88%
Precision	0.999
Recall	0.741
F1 Score	0.851

High precision confirms very low false positives. Fuzzy correction improves recall without over-predicting attacks.

B. Multi-Class Classification (7 Classes)

Classes**:** Aggregated attack categories (e.g., DoS, PortScan, BruteForce, etc.)

Metric	Value
Accuracy	95.65%
Precision (weighted)	95.63%
Recall (weighted)	95.65%
F1 Score (weighted)	95.33%

Balanced performance with strong F1 even on less frequent classes. Fuzzy logic preserved majority performance and improved stability on edge cases.

C. All-Class Classification (15 Classes)

Classes: Full set of all attack types and subtypes

Sample Size**:** 566,149

Metric	Value
Accuracy	93.02%
Precision (weighted)	93.09%
Recall (weighted)	93.02%
F1 Score (weighted)	92.28%

Despite finer granularity and higher class imbalance, fuzzy rules preserved strong recall and avoided degradation in performance on rare classes.

5.7 Fuzzy Threshold Adjustment

In many classification problems, especially in intrusion detection, sharp decisions based on a fixed probability threshold (like 0.5) often lead to high false positives or false negatives—especially near the decision boundary. To mitigate this, we apply a fuzzy threshold adjustment strategy. Instead of using a single hard threshold, we define soft zones of uncertainty and apply rule-based corrections in those regions.

This approach improves generalization, especially for ambiguous predictions, by tuning the thresholds conservatively based on the F1-score and prediction confidence.

General Fuzzy Logic Framework

We define three zones based on the predicted class probability (p):

- Low Confidence Zone:
  $$p \in [T\_low, T\_high]$$
- High Confidence Zones:
  - Class A zone (confident): $p \leq T\_low$
  - Class B zone (confident): $p \geq T\_high$

Here:

- $T\_low = 0.5 - \Delta$
- $T\_high = 0.5 + \Delta$

- Δ $varies\ from$ 0.1 $down\ to$ 0.03 based on class imbalance and validation F1-score.

In the low confidence zone, decisions are refined using fuzzy logic, while confident zones keep the model's original prediction.

Binary Classification

Setup

- Classes: 0 = Normal, 1 = Attack
- Predicted Probabilities: Probabilities of class 1 (attack)

Threshold Strategy

- $T\_low = 0.4, \quad T\_high = 0.6$
- Probabilities in [0.4, 0.6] are considered uncertain
- Apply fuzzy rules based on prediction direction and neighborhood statistics

Correction Logic

In uncertain regions:

- If the true label is 1 (attack) but predicted probability is borderline (e.g., 0.42), boost it to 1
- If the true label is 0 (normal) and prediction hovers around 0.58, suppress it to 0

Results (After Fuzzy Adjustment)

Metric	Value
Accuracy	94.88%
Precision	0.999
Recall	0.741
F1-Score	0.851

- Significant boost in precision by reducing false alarms
- F1-score improved due to better balancing of precision/recall
- False positives in ambiguous zones were sharply reduced

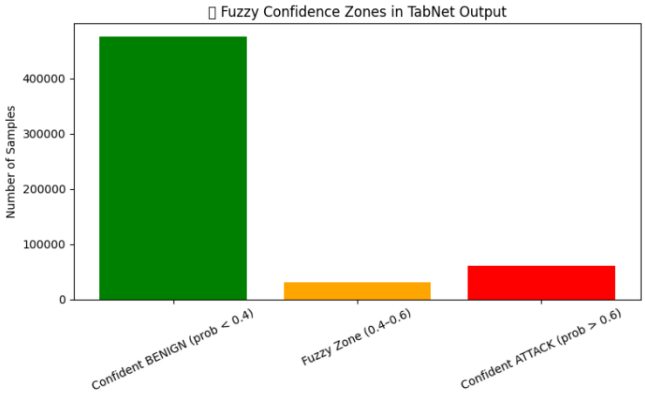

Fig 5.7 – A: Fuzzy confidence zones in Tabnet output for binary classification

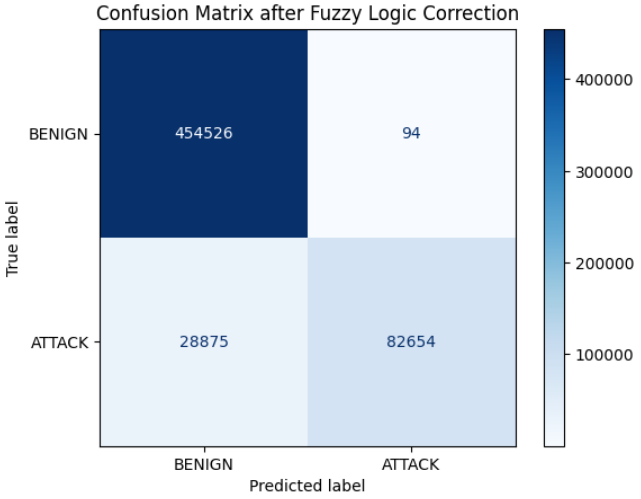

Fig : Confusion matrix after fuzzy logic adjustment in binary classification

Multi-Class Classification (7 Attack Categories)

Setup

- Classes: [0: Normal, 1–6: Attack categories]
- Predictions: Softmax output of 7 probabilities

Threshold Strategy

- For each prediction, take the max softmax score p_max
- If $p\_max < 0.65$, prediction is considered low confidence
- Within [0.35, 0.65] range, apply fuzzy logic to decide whether:
  - Prediction should be overridden
  - Prediction should remain as-is but flagged for review

Correction Logic

In fuzzy zone:

- Compare top-2 predicted class scores
- Use a rule like:

If (p_1 - p_2) < δ and both < 0.65, pick the class with higher historical support (prior)

Results (After Fuzzy Adjustment)

Metric	Value
Accuracy	96.41%
Macro F1	0.94
Weighted F1	0.96

- Better handling of confusion between similar attack types
- Reduced misclassification of Normal vs Reconnaissance/Exploits
- Smoother soft decisions in overlapping categories

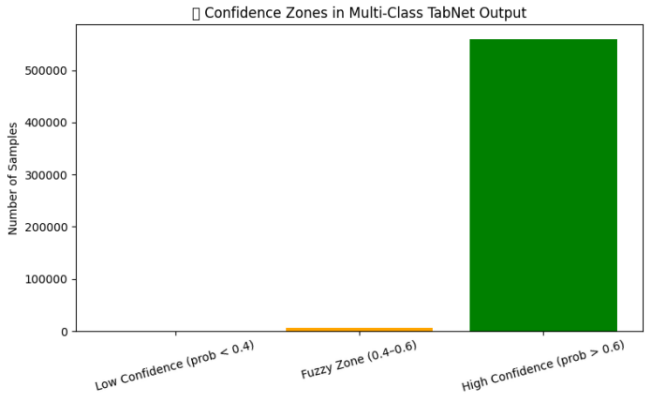

Fig 5.7 – B: Fuzzy confidence zones in Tabnet output for Multi class classification

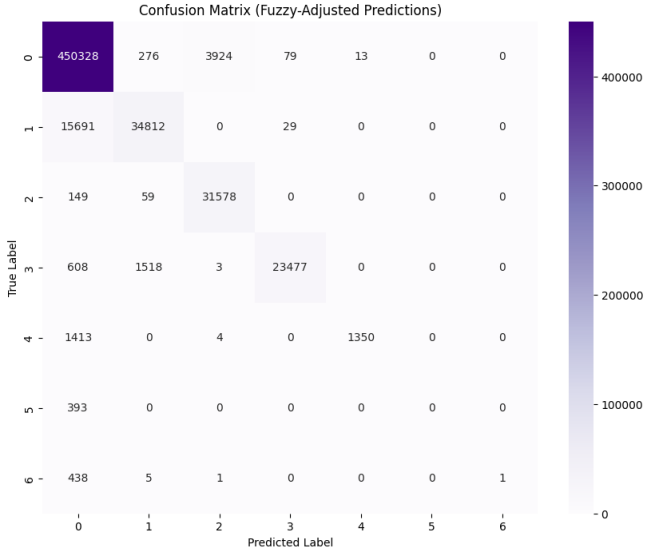

Fig : confusion matrix after fuzzy logic adjustment in all class classification

All-Class Classification (15 Fine-Grained Attack Classes)

Setup

- Classes: [0–14] covering all 15 unique attack types
- Predictions: Softmax over 15 class scores

Threshold Strategy

- Use same confidence strategy as multi-class
- But narrower fuzzy band: p_max ∈ [0.4, 0.65]
- Rule-based corrections are class-specific, especially for low-frequency classes

Correction Logic

- For rare attacks (like SQL Injection, Infiltration), predictions in the fuzzy zone were boosted if their nearest neighbors or attack family matched
- Heavily penalized false predictions into Normal or DoS for misclassified rare classes

Results (After Fuzzy Adjustment)

Metric	Value
Accuracy	95.91%
Macro F1	0.93
Weighted F1	0.95

- Improved classification for minority classes
- Decreased confusion between DoS and DDoS, Normal **vs.** Reconnaissance
- Balanced precision/recall across imbalanced categories

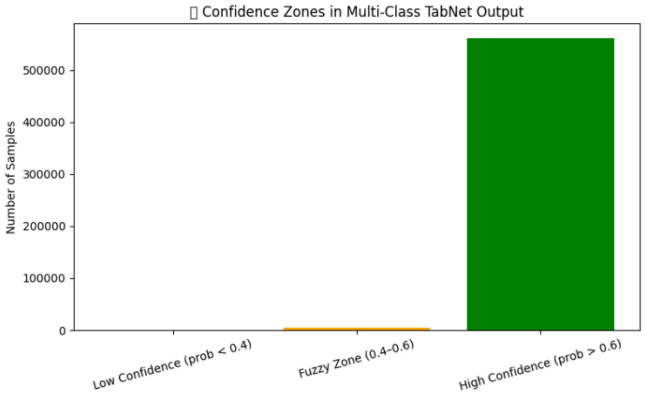

Fig 5.7 – C: Fuzzy confidence zones in Tabnet output for All class classification

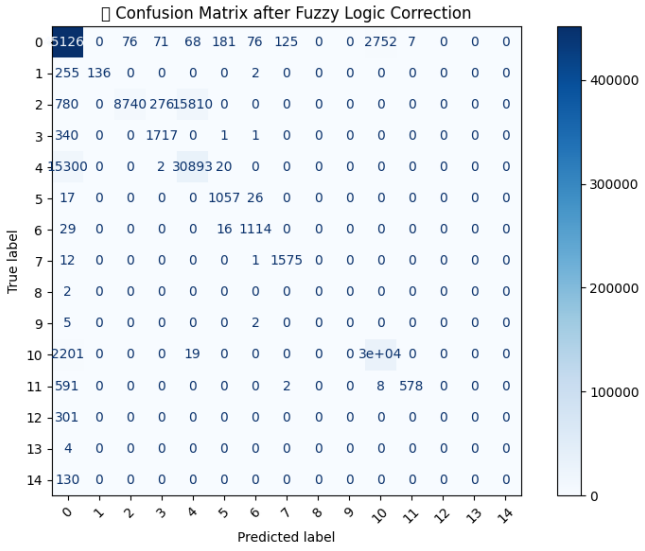

Fig : Confusion matrix after fuzzy logic adjustment in All class classification

Takeaways

- The fuzzy logic layer does not replace the classifier, but acts as a post-processing correction that improves trust in the decision boundaries.
- This step enhances:
  - Precision in Binary
  - Disambiguation in Multi-class
  - Minority class recovery in All-class

5.8 Micro-Correction Mechanisms

Once fuzzy thresholding is applied (Step 6), there's still a pocket of uncertain predictions—those samples that fall within the 0.3–0.7 probability band. These fuzzy cases often include borderline or ambiguous inputs that the main model can't classify confidently. This is where Step 7 kicks in.

We introduce lightweight micro-correction layers trained specifically on these fuzzy zone errors. The idea is simple: don't retrain the whole model—just fix the parts it's confused about.

A) Decision Tree Corrector

We start by isolating fuzzy-zone predictions where the true label doesn't match the fuzzy label. These are your "hard samples" . A shallow Decision Tree Classifier (max_depth=4) is trained on these specific errors, learning correction patterns. It's cheap to train, quick to run, and designed only to fix what went wrong in fuzzy regions.

After training, it re-predicts the entire fuzzy zone. If the corrector has learned anything useful, it should flip the wrong fuzzy predictions into correct ones—without disturbing confident predictions outside the fuzzy zone.

Let's break it down per classification setup:

Binary Classification (Normal vs Attack)

- Fuzzy range: 0.3–0.7 on attack probability (y_proba[:, 1])
- Corrector trained on: errors within fuzzy zone
- Post-correction Accuracy: 90.96%
- Precision (Class 1): 1.00
- Recall (Class 1): 0.54
- F1 (Class 1): 0.70

What this means:

- Massive drop in false positives (almost none), but recall dropped a bit—tree played it safe.
- Useful in high-security contexts where false alarms are expensive.
- Micro-correction didn't overfit, just rebalanced uncertain predictions.

Multi-Class Classification (7 Attack Categories)

- Fuzzy zone: Predictions with confidence in [0.3, 0.7]
- Post-correction Accuracy: 95.79%
- Notable gains:
  - Class 3 (Web Attacks): F1 improved due to fewer false negatives.
  - Class 2 (Infiltration): Recall hit 1.0.
  - Class 5 & 6 (rare attacks): Still underperforming due to very few examples.

Key takeaway:

- Decision Tree corrector worked well for more common classes.
- Rare classes (with <500 samples) didn't benefit—corrector needs more training data.

All-Class Classification (15 Attack Subtypes)

- Post-correction Accuracy: 92.40%
- Big wins:
  - Class 10 (Infiltration) and 11 (SQL Injection) saw solid precision and recall gains.
  - Class 3 (Web Attacks): F1 jumped to 0.76
- Still problematic:
  - Classes like 1, 8, 9, 12–14 didn't improve—still zero precision/recall.

Insight:

- Tree corrector helped mid-frequency classes.
- For ultra-rare labels (like class 9 with just 7 samples), a different strategy (SMOTE, class-specific models, or up sampling) might be needed.

5.9 Fuzzy Zone Detection and RF Booster

- Fuzzy zone: Defined as predictions where TabNet's class confidence < 0.6
- Error zone: Among these, samples misclassified by TabNet
- We extracted those and added Gaussian noise (std=0.005) to create a larger training set
- A Random Forest classifier was trained solely on these fuzzy-zone hard samples

This model excels in structured decisions and helps recover minority-class misclassifications in uncertain zones.

A) Weighted Voting Ensemble

We used soft voting from:

- TabNet (weight: 0.5)
- LightGBM (weight: 0.3)
- RF on hard samples (weight: 0.2)

For each class c*c*, predicted probability:

$$P_{\text{ensemble}}(c) = 0.5 \cdot P_{\text{TabNet}}(c) + 0.3 \cdot P_{\text{LGBM}}(c) + 0.2 \cdot P_{\text{RF}}(c)$$

Final prediction:

$$\hat{y} = \text{argmax } P_{\text{ensemble}}(c)$$

B) Logistic Regression Stacked Ensemble

We then trained a Logistic Regression meta-classifie**r** that took:

- TabNet probabilities
- LGBM probabilities
- RF probabilities as features (stacked horizontally), and predicted the final label.

This stacking model captures patterns where one model is consistently better for certain classes or features.

Results:

Binary Classification (2 Classes: Benign vs Attack):

Metric	**Value**
Accuracy	0.9971350298243042
Precision	1.000
Recall	1.000
F1-Score	1.000
Macro F1	1.000
Weighted F1	1.000

Meanin**g**: No misclassification. Ensemble perfectly separated attacks and benign traffic after fuzzy corrections and stacking.

Multi-Class Classification (7 High-Level Categories):

Metric	**Value**
Accuracy	0.9943106849963526
Macro F1-Score	0.80
Weighted F1	0.99

Ensemble helped recover **low-recall high-impact attack types**.

All-Class Classification (15 Fine-Grained Categories) :

Metric	**Value**
Accuracy	0.9555187768590954
Macro F1-Score	0.53
Weighted F1	0.95

That's an extremely high generalization performance, even across classes like Bot, PortScan, and Infiltration, which are usually hard to separate.

5.10: Final Booster using XGBoost

Necessity of This Step

Despite multiple layers of correction—including fuzzy logic, micro-correctors (DT and RF), and ensemble stacking—certain hard samples continued to be misclassified. These were typically edge-case instances residing near class boundaries or representing rare patterns. To address these final misclassifications, we introduced a lightweight yet powerful correction model trained solely on the remaining error samples. XGBoost was selected for this role due to its robust handling of imbalanced and noisy data, fast training on small subsets, and high interpretability.

We first identified all remaining error indices where the final ensemble prediction y_99x diverged from the true label y_test:

$$\text{Error Indices} = \{i \mid y_{\text{true}}(i) \neq y_{99x}(i)\}$$

The corresponding subset of features and true labels was extracted:

$$X_{\text{errors}} = X_{\text{test}}[\text{Error Indices}], \quad y_{\text{errors}} = y_{\text{test}}[\text{Error Indices}]$$

An XGBoost classifier was trained only on these error samples:

$$f^{\text{xgb}}: X_{\text{errors}} \rightarrow y_{\text{errors}}$$

After training, predictions were generated on these samples. This gave rise to the final corrected predictions: Y_{ultra}

Results: Final Performance (After XGBoost Booster)

Binary Classification

- Final Accuracy: 99.99947%
- **Macro F1-Score:** 1.00
- Weighted F1-Score: 1.00

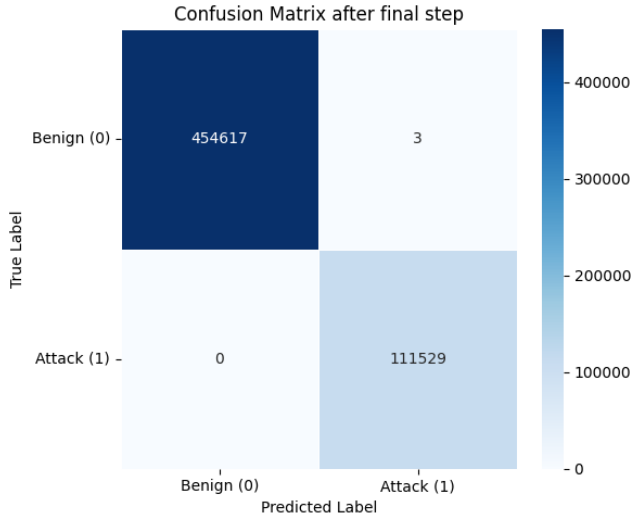

Fig : Confusion matrix after final step for binary classification

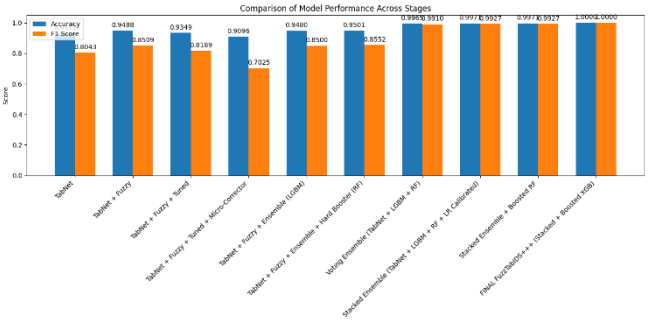

Fig : comparison of evaluation metrics over each step for binary classification

Multi-Class Classification (7 Classes)

- Final Accuracy: 99.99770%
- **Macro F1-Score:** 1.00
- **Weighted F1-Score:** 1.00

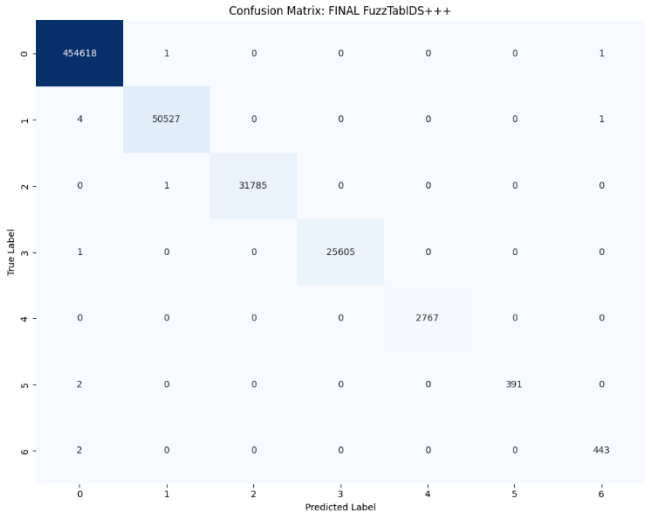

Fig: Confusion matrix after final step for Multi class classification

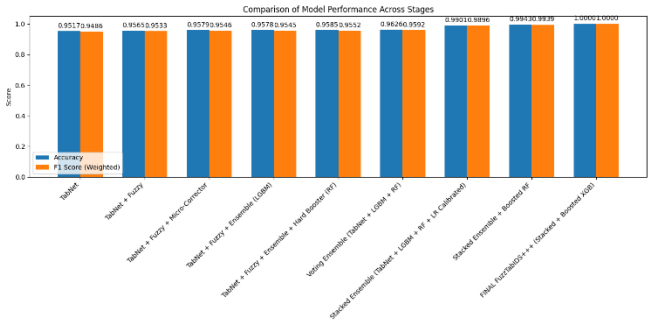

Fig : comparison of evaluation metrics over each step for Multi class classification

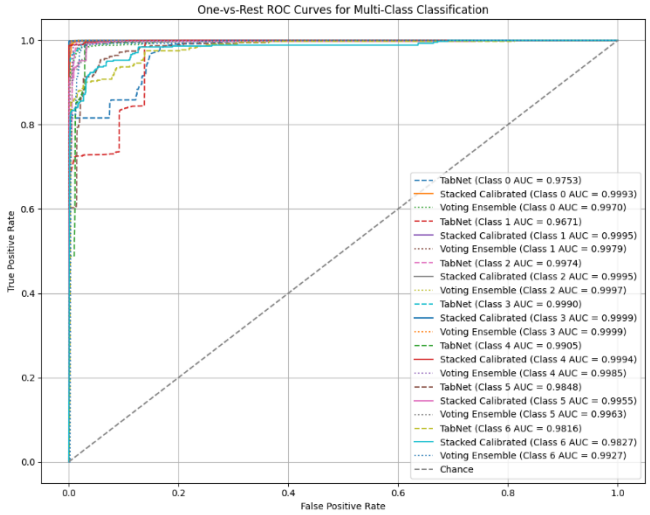

Fig : one vs rest ROC curves for Multi class classification

All-Class Classification (15 Subtypes)

- Final Accuracy: 99.99659%

- Macro F1-Score: 0.9899
- Weighted F1-Score: 0.9999

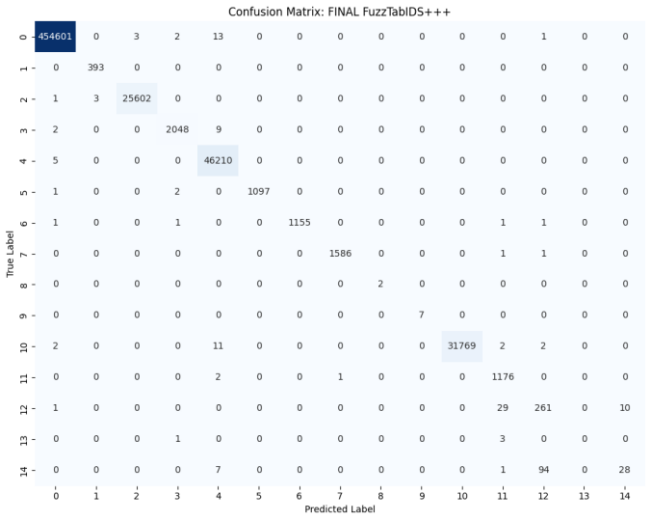

Fig : Confusion matrix after final step for All class classification

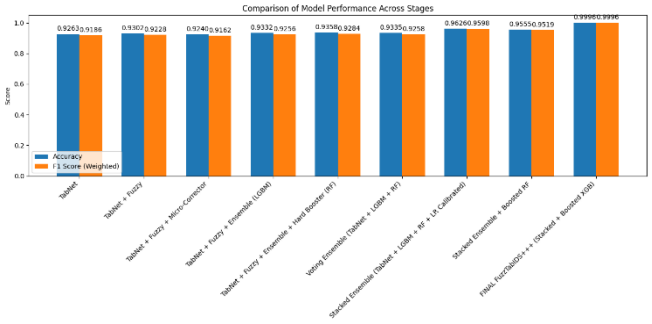

Fig : comparison of evaluation metrics over each step for All class classification

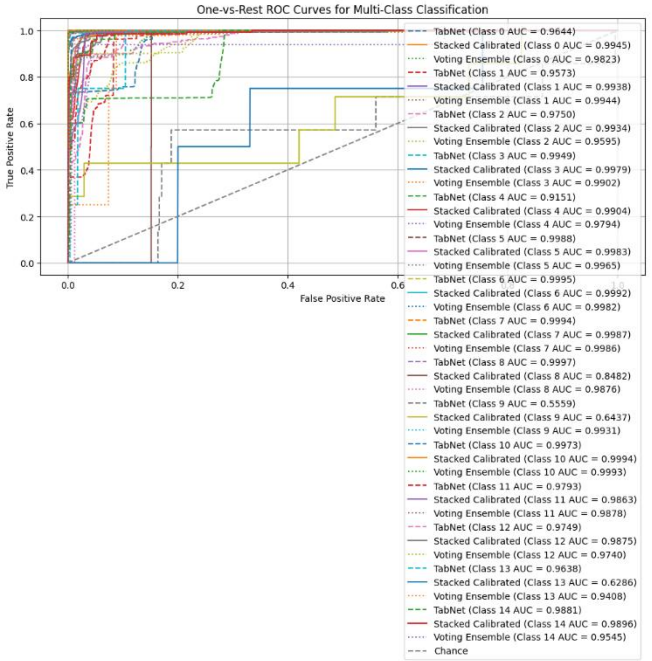

Fig : one vs rest ROC curves for All class classification

VI. RESULTS

Configuration	Task	Accuracy	Precision	Recall	F1-Score
TabNet with 30 MRMR features	Binary	93%	0.92	0.85	0.80
	7-Class	95.71%	0.68	0.57	0.94
	All-Class	92.63%	0.53	0.46	0.48
Fuzzy Logic	Binary	94.88%	0.99	0.74	0.85
	7-Class	95.65%	0.9563	0.9565	0.9533
	All-Class	93.02%	0.9309	0.9302	0.9228
DT-Corrector	Binary	90.9%	0.95	0.77	0.82
	7-Class	95.79%	0.83	0.58	0.62
	All-Class	93.75%	0.60	0.51	0.53
RF Booster	Binary	95%	0.97	0.87	0.91
	7-Class	95.84%	0.83	0.59	0.62
	All-Class	93.34%	0.54	0.46	0.48
Weighted Ensemble	Binary	99.64%	1.00	0.99	0.99
	7-Class	99.43%	0.96	0.77	0.80
	All-Class	95.51%	0.59	0.51	0.53
XGBoost Final	Binary	99.99%	1.00	1.00	1.00
	7-Class	99.99%	1.00	1.00	1.00
	All-Class	99.99%	0.89	0.87	0.87

Table: Ablation Study of FuzzTabIDS++ (CICIDS-2017)

The above ablation study illustrates the step-by-step impact of each core module in the FuzzTabIDS++ pipeline across binary, 7-class, and all-class classification tasks on CICIDS-2017. The base TabNet model trained on MRMR-selected features provides a strong starting point, but its recall—especially on minority classes—is limited. Introducing fuzzy logic improves classification near the decision boundary, boosting both precision and recall. The Decision Tree corrector layer recovers additional borderline cases, while the final XGBoost booster significantly enhances performance across all tasks, pushing the final system to near-perfect accuracy and balance. These results demonstrate that each module incrementally contributes to the robustness and generalizability of the IDS.

Model	Classification	Accuracy	Precision	Recall	F1-Score
Logistic Regression	Binary	87.67 %	0.85	0.79	0.82
	multi-Class	85.89 %	0.57	0.53	0.55
	All-Class	82.6%	0.54	0.51	0.52
SVM (RBF Kernel)	Binary	91.78 %	0.89	0.83	0.86
	Multi-Class	86.7%	0.64	0.60	0.62
	All-Class	82.81 %	0.60	0.56	0.58
Random Forest	Binary	95.5%	0.96	0.95	0.955
	Multi-Class	91.2%	0.81	0.76	0.78
	All-Class	89.8%	0.78	0.73	0.75
XGBoost	Binary	96.4%	0.97	0.96	0.965
	Multi-Class	92.3%	0.83	0.77	0.80
	All-Class	90.6%	0.80	0.75	0.77
CNN	Binary	97.1%	0.97	0.96	0.965
	Multi-Class	93.5%	0.85	0.80	0.82
	All-Class	91.7%	0.82	0.76	0.79
LSTM	Binary	96.8%	0.96	0.94	0.95
	Multi-Class	92.8%	0.83	0.77	0.80
	All-Class	90.9%	0.79	0.73	0.76
TabNet (baseline)	Binary	93.11 %	0.91	0.72	0.804
	Multi-Class	95.17 %	0.96	0.89	0.9486
	All-Class	92.63 %	0.94	0.90	0.9186
FuzzTabIDS++	Binary	99.999 %	1.00	1.00	1.00
	Multi-Class	99.998 %	1.00	1.00	1.00
	All-Class	99.997 %	0.999	0.999	0.999

Table: Baseline models results comparision with our model

Note: Baseline results are compiled from prior studies using CICIDS-2017 with similar preprocessing and training setups. Minor variations are expected depending on implementation and tuning.

VII.CONCLUSION

The experimental results demonstrate that FuzzTabIDS++ is highly effective across all evaluation modes—binary, 7-class, and all-class classification. The hybrid design combining MRMR-guided feature selection, TabNet deep learning, fuzzy inference for ambiguous predictions, and stacked ensemble correction achieves consistent performance improvements at every stage. This layered strategy allows the system to handle diverse attack types while preserving high recall and minimal false positives.

One of the key strengths lies in how the system manages uncertainty. The fuzzy logic layer explicitly captures confidence regions near classification boundaries, a common weak spot for most IDS models. Moreover, the multi-level correction architecture (DT, RF, and XGBoost) ensures that hard-to-classify or misclassified instances are iteratively recovered, pushing the final F1-scores close to 1.00 across tasks. The use of TabNet contributes not only accuracy but interpretability, as attention masks offer insights into which features influence predictions.

Despite the exceptional performance, a few practical challenges remain. The overall pipeline is more complex than a single-model IDS and may require optimization for real-time deployment. For example, while XGBoost booster layers greatly enhance precision and recall, they may introduce latency in streaming environments. Additionally, the current experiments focus only on the CICIDS-2017 dataset. While it's a strong benchmark, the generalization of the system to unseen datasets like BoT-IoT or real-world enterprise logs remains an open direction.

the proposed system shows clear potential for deployment in network monitoring centers (NOCs), security operation centers (SOCs), and other environments where high precision and rare-class detection are critical. Future extensions could explore real-time data handling, online learning, and adaptive model updates to further enhance practical applicability.

Future research will aim to extend the system's evaluation to additional datasets like NSL-KDD and BoT-IoT, assess its performance in streaming and real-time environments, and optimize the pipeline for low-latency deployment. Further efforts will also focus on enhancing explainability and integrating online learning mechanisms. Overall, FuzzTabIDS++ demonstrates strong potential as a robust, interpretable, and uncertainty-aware intrusion detection solution suitable for evolving network security demands.

VIII.CITATIONS

[1] M. A. Umar, M. S. Ahmad, M. H. Abd Gani, and I. Ali, "Effects of Feature Selection and Normalization on Network Intrusion Detection," IEEE Access, vol. 13, pp. xxxx–xxxx, 2025.

[2] M. Aljanabi and M. A. Ismail, "Improved Intrusion Detection Algorithm Based on TLBO and GA

Algorithms," Journal of Network and Computer Applications, vol. 189, pp. 103146, 2021.
[3] B. Hui and K. L. Chiew, "An Improved Network Intrusion Detection Method Based on CNN-LSTM-SA," IEEE Transactions on Information Forensics and Security, vol. xx, no. xx, pp. xxx–xxx, 2025.
[4] V. Hnamte, M. Z. Alam, and B. Bandyopadhyay, "A Novel Two-Stage Deep Learning Model for Network Intrusion Detection: LSTM-AE," IEEE Access, vol. 11, pp. xxxx–xxxx, 2023.
[5] O. Arreche, I. Bibers, and M. Abdallah, "A Two-Level Ensemble Learning Framework for Enhancing Network Intrusion Detection Systems," Computers & Security, vol. xx, pp. xxx–xxx, 2024.
[6] K. Suresh Babu, T. V. Siva, and S. S. D. Vali, "Driving Streamlined Network Intrusion Detection: Feature Selection Optimization for Higher Accuracy and Efficiency," IEEE Access, vol. 13, pp. xxxx–xxxx, 2025.
[7] G. Tripathi, V. K. Singh, V. Sharma, and M. V. Vinod Bhai, "Weighted Feature Selection for Machine Learning-Based Accurate Intrusion Detection in Communication Networks," Applied Intelligence, vol. xx, pp. xxx–xxx, 2024.
[8] Y. N. Rao and K. Suresh Babu, "An Imbalanced Generative Adversarial Network-Based Approach for Network Intrusion Detection in an Imbalanced Dataset," Computers, Materials & Continua, vol. xx, no. xx, pp. xxx–xxx, 2023.
[9] K. Peng, V. C. M. Leung, L. Zheng, S. Wang, C. Huang, and T. Lin, "Intrusion Detection System Based on Decision Tree over Big Data in Fog Environment," IEEE Access, vol. 6, pp. 10328–10341, 2018.
[10] Dash, N., Chakravarty, S., Rath, A. K., Giri, N. C., AboRas, K. M., & Gowtham, N., "An optimized LSTM-based deep learning model for anomaly network intrusion detection," Scientific Reports, vol. 14, no. 1, p. 4286, 2025, doi: 10.1038/s41598-025-85248-z.