

Evaluation of classical and deep learning models for network intrusion detection

Nate Huizar, *Purdue University Northwest*, huizarn@purdue.edu

Tim Cantu, *Purdue University Northwest*, tcantu@purdue.edu

George Stefanek, *Purdue University Northwest*, stefanek@purdue.edu

Abstract

Network Intrusion Detection Systems utilizing flow-based metadata have difficulty in detecting application layer web attacks such as SQL-injection and cross-site scripting. Standard machine learning models suffer from base rate fallacy, reducing detection reliability. This research evaluates both classical and deep learning algorithms, including Logistic Regression, Random Forest, XGBoost, Multilayer Perceptron, and 1D-CNN to determine which performs best on detecting various types of network intrusion attacks when trained on imbalanced datasets. To overcome detection failures, an a) iterative machine learning methodology and b) a structural class aggregation technique that groups fragmented application-layer threats into a single macro-class are used. These approaches act as a practical workaround for the limitations associated with flow-based network data thereby making it more suitable within a security operations center. It was found that the XGBoost model trained on aggregated data provided the best performance on detecting multiple attack types including minority-categories as compared to 1D-CNN or MLP models. Also, it was found that cost-sensitive optimization using a weight-penalty strategy was successful in helping the model to prioritize minority-class detection reaching a 99% recall rate. Finally, it was found that macro-class aggregation was an effective method for increasing the signal-to-noise ratio for sparse data in imbalanced datasets.

Keywords: Machine Learning, Intrusion Detection, Logistic Regression, Random Forest, XGBoost, CNN

Introduction

The reliance on Network Intrusion Detection Systems (NIDS) as part of today's overall cybersecurity strategy continues to grow as networks become larger with steadily increasing amounts of network traffic. A primary function of a NIDS is to act as a real-time filter which separates benign traffic from malicious traffic. While NIDS have proven to be capable of detecting volumetric attacks such as DDoS and Port Scans, they have failed at detecting stealthy application layer attacks such as SQL-injection and cross-site scripting (XSS).

This research compares the performance of both classical and deep learning models in identifying these types of threats utilizing the CIC-IDS-2017 dataset. The CIC-IDS-2017 dataset utilizes flow-based network data such as packet duration, packet length, and flags. Therefore, unlike deep packet inspection, these algorithms identify the malicious payload by examining the "envelope" of the network traffic rather than reading the content inside. This limitation creates a substantial problem like extreme class imbalance with standard machine learning architectures. Malicious web attacks account for less than 0.5 percent of all recorded network traffic within the scope of the CIC-IDS-2017 dataset (Sharafaldin, et al., 2018). Standard machine learning algorithms suffer from a common phenomenon called base rate fallacy that

causes them to rely on global accuracy by always predicting the most likely benign majority class (Garcia-Teodoro, 2009). Therefore, despite their statistical success and the achievement of accuracy greater than 99%, these models are poor at detecting minority classes of threats that they were designed to detect (Zhao, 2023).

Statement of the Problem

Network Intrusion Detection Systems utilizing flow-based metadata have difficulty in identifying some types of web attacks like SQL injection and cross-site scripting due to models being trained on datasets with extreme class imbalance. This research presents an iterative machine learning methodology to overcome these detection failures along with a structural class aggregation technique to group fragmented application-layer threats into a single "Web Attack" macro-class. These two concepts together can be utilized as workarounds for the mathematical limitations associated with flow-based network data, thereby creating an operational model more suitable for implementation within a security operations center (SOC).

Literature Review

The Fundamental Shift in NIDS Benchmarking

The effectiveness of Network Intrusion Detection Systems is limited by the quality of their training data. Research has relied on the KDD Cup 99 and NSL-KDD datasets for many years. However, Sharafaldin et al. (2018) discovered weaknesses in these legacy benchmark datasets, citing their failure to represent modern network traffic patterns. Sharafaldin's introduction of the CIC-IDS-2017 dataset provides researchers with a dataset that includes bidirectional network flows and over 80 metadata features. This dataset also includes a wide range of intrusions thus becoming an excellent and robust dataset for machine learning algorithms.

Theoretical Foundations of Ensemble Learning

The use of ensemble learning that combines multiple models to improve predictive performance has been used to map the nonlinear complexities of network traffic (Breiman, 2001). Breiman showed that grouping multiple independent trees into a "random forest" reduces variance and counteracts the overfitting problem. This approach allows for a nonlinear baseline that improves accuracy over simple linear models like logistic regression. Improving on this approach, Chen and Guestrin (2016) introduced the XGBoost model, a highly scalable gradient boosting system that employs "boosting" to iteratively correct the errors of previous trees. XGBoost provides improved precision and recall on fitted, tabular data, making it an important model for identifying intrusions buried beneath benign traffic as in the CIC-IDS-2017 dataset.

Deep Learning

The field of deep learning for network security represents a method to deal with the huge amounts of complexity in network metadata. Mohammadpour et al. (2022) state that using CNNs allows one to look at data completely differently than with traditional machine learning or other AI techniques. The use of these advanced deep learning models is limited since the models cannot be applied to common industrial benchmarks like CIC-IDS-2017 and have an accuracy paradox first described by Zhao et al. (2023). Zhao showed that while most of the top performing models can achieve almost perfect accuracy (i.e., nearly 100%) on benchmarking datasets, those same models perform poorly in detecting minority class attack vectors. That gap between reported accuracy and real-world detection capability results in what can be referred to as "The Accuracy Illusion". In other words, the reported performance metrics for a model

mask its' inability to detect hard-to-spot attacks. To decrease the gap between reported performance and actual real-world detection capability, researchers have transitioned from developing generalized classifiers to developing specialized ones. For example, a recent study conducted by Osa (2024) indicated that developing a NIDS based on deep learning techniques requires significantly more than just developing a classifier architecture. Instead, the NIDS needs to include the ability to recognize complex and nonlinear signatures associated with various intrusion types.

While ensemble methods mostly dominate tabular data analysis, Kim et al. (2020) was one of the first to explore the use of deep learning algorithms in network intrusion detection. Kim's research focused on the capabilities of models such as Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), to perform automated feature extraction. Combined with layers of activation functions, like ReLU, these models can uncover hidden mathematical relationships between network features that labor intensive manual engineering might miss. A drawback of current research also highlights a significant challenge; while deep learning models excel at unstructured data, like images, it is common for these models to struggle with high dimensional, tabular data such as that in network flows, resulting in much higher false positive rates compared to tree-based ensemble methods.

Building upon this foundation, research from Farhan et al. (2025) and Ajdani (2025) indicate that the next generation of network intrusion detection systems will need to employ cost sensitive training methodologies. Their work indicates that the utility of a model is entirely dependent on its ability to correctly identify rare but potentially dangerous anomaly events. The focus of recent research activity is shifting towards the design of systems that place a higher value on recall sensitivity to rare anomalous activity rather than on deceptive global accuracy measures. Additionally, recent literature by Almuhanha et al. (2025) presents a hybrid anomaly-based Network Intrusion Detection System that integrates multiple machine learning and deep learning algorithms, including XGBoost, Random Forest, Graph Neural Networks (GNN), Long Short-Term Memory (LSTM) networks, and Autoencoders to achieve 100% accuracy on their dataset.

The Class Imbalance Challenge

Even with the dominance of highly advanced algorithms, the base rate fallacy is still the most significant challenge in intrusion detection. Panigrahi and Borah (2018) performed an exhausting analysis of the CIC-IDS-2017 dataset identifying an extreme sparsity problem among critical minority classes. Their research demonstrates that when certain attacks like SQL injections or cross-site scripting represent less than 0.1% of the total dataset, classical machine learning models mathematically concentrate on global accuracy over the detection of web attack types that occur at lower frequencies. Consequently, the models may achieve 99% accuracy but still fail to detect the threats they were designed to detect. Their findings justify moving past global accuracy as the primary metric. Harish et al (2026) propose a novel hybrid intrusion detection methodology designed to mitigate data imbalance issues which uses advanced deep learning techniques and optimized characteristic fusion models to achieve a 96.43% accuracy, making it desirable for high-traffic environments.

Mitigation Strategies and Structural Pivots

To address the failures caused by these class imbalances, researchers like to choose between data level and algorithmic level solutions. Techniques like SMOTE (Synthetic Minority Over-Sampling Technique) are common, however research warns that synthetic data can present "imagined" patterns that typically do not exist in real world traffic relevancy (He & Garcia, 2009). A rising trend favors cost sensitive learning, which modifies the algorithm's loss function to apply a penalty weight to the minority class misclassifications. Studies also hint that breakup of similar threats such as web-based attacks can reduce the detection signal. These findings support the use of macro-class aggregation which group logically

similar categories/attacks into a single macro category to provide a greater sample size without reducing accuracy (Panigrahi & Borah, 2018).

Methodology

The technologies used in this research included Python 3.7 and Jupyter Notebooks. Machine learning performance libraries for data management included Pandas and NumPy and the Scikit-Learn library was used for preprocessing and classical modeling. Deep learning models were created using TensorFlow and the Keras frameworks.

Data Preprocessing and Feature Engineering

The CIC-IDS-2017 dataset was used in this study and can be found on Kaggle. This dataset went through multistage transformations to make it more suitable for use with both classical and deep learning models. The preprocessing steps included the following:

Cleaning and Filtration

To address the comparative detection profiles of low-prevalence application-layer vectors against high-volume volumetric events, a specific focus area was isolated from the full 2.83 million records of the CIC-IDS-2017 benchmark. Three distinct PCAP-generated CSV files were selected and combined: Friday-WorkingHours-Afternoon-DDos.csv, Friday-WorkingHours-Afternoon-PortScan.csv, and Thursday-WorkingHours-Morning-WebAttacks.csv. This targeted aggregation produced an initial dataset pool of 682,038 instances. Within this subset, rows containing NaN (Not a Number) values or infinity notations, generated by calculation constraints during flow metadata conversion, were filtered out to eliminate training compilation errors. The final distribution of clean target rows across categories is highlighted in Figure 1.

```
-----  
Total Rows: 682038  
Total Features: 79  
Label  
BENIGN                                393029  
PortScan                             158804  
DDoS                                  128025  
Web Attack 🔒 Brute Force              1507  
Web Attack 🔒 XSS                      652  
Web Attack 🔒 Sql Injection            21  
Name: count, dtype: int64
```

Figure 1. Dataset Record Breakdown by Network Attack Category

Label Encoding

String-based attack categories were converted into integers. For the last parts of the run, a custom mapping function was implemented to aggregate fragmented labels into a broader macro class labeled “Web Attack” as shown in Figure 2.

```
Label Mapping for Research Paper:  
{'BENIGN': 0, 'DDoS': 1, 'PortScan': 2, 'Web Attack 🔒 Brute Force': 3, 'Web Attack 🔒 Sql Injection': 4, 'Web Attack 🔒 XSS': 5}
```

Figure 2. Label Mapping

Feature Selection

The CIC-IDS-2017 dataset has over 80+ features of which many features are highly correlated. The Random Forest Classifier was used to help pick features. The top 15 features extracted are shown in Figure 3. The reason for choosing 15 features was to prevent overfitting and that it was roughly 20% of all the features in the dataset.

Feature Normalization

Feature normalization was applied to alleviate the dominance of features with large numerical ranges (e.g., total backward packets, etc.). A scaling technique was employed to transform each feature to have a mean of zero and a standard deviation of one, thereby ensuring that all features contribute equally to the model's training process rather than one overpowering the other.

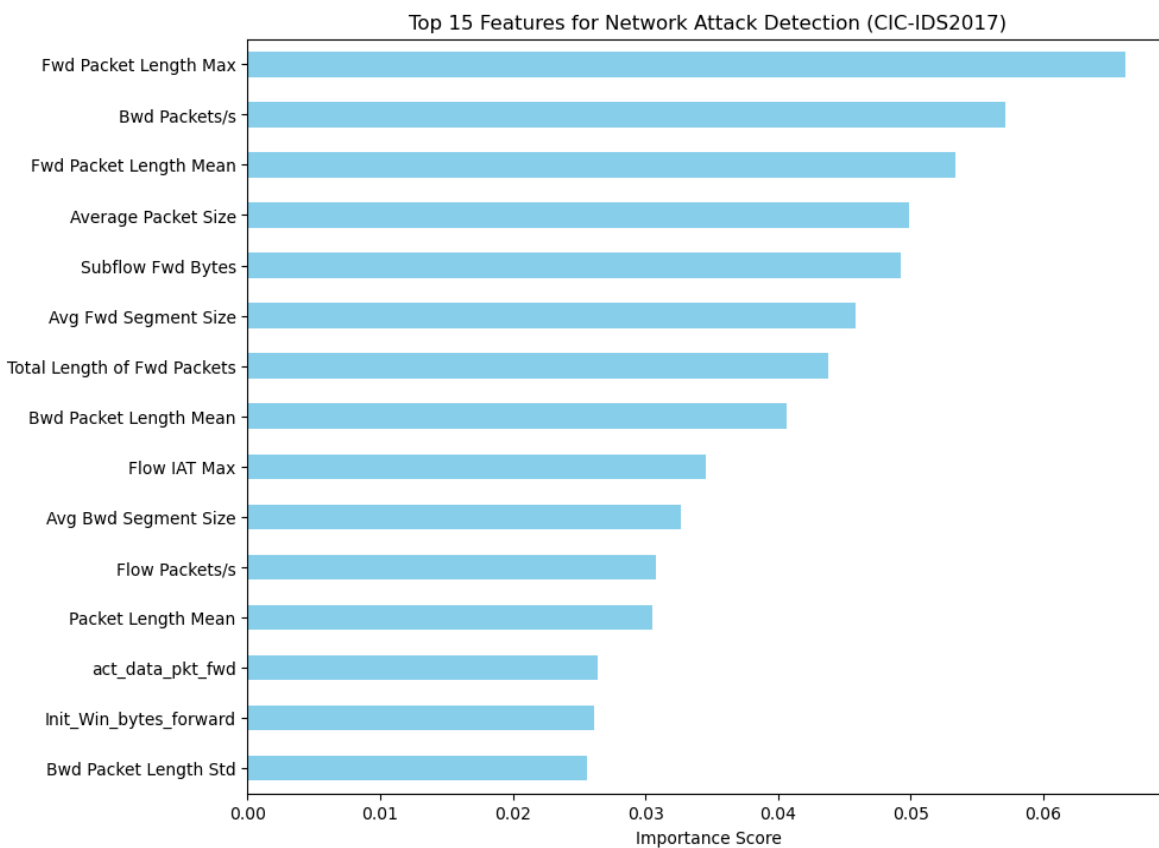


Figure 3. Top 15 Features Extracted from the CIC-IDS-2017 Dataset

Train-Test Split

Prior to model training, the aggregated focus dataset was partitioned into an 80% training split and a 20% testing split via a stratified holdout approach. For deep learning architectures like MLP and 1D-CNN, an internal 20% validation split was used during training to monitor epoch convergence and prevent overfitting. Models used standard baseline hyperparameter configurations (e.g., 100 estimators for Random Forest). To preserve strict isolation and prevent data leakage, feature normalization was conducted sequentially: Scikit-Learn's StandardScaler computed scaling parameters (mean and variance) exclusively from the training partition data matrix via a `.fit_transform()` invocation. These locked parameters were then projected to the evaluation test split via the `.transform()` method.

Hierarchical Model Architectures

Baseline Linear Classifier

Using L2 regularization, which is a weight decay process, a Logistic Regression model was trained as a linear baseline. This model provided a reference point for evaluating the performance achieved by more complex, nonlinear classification architectures. The training and testing data was scaled using the SKLearn “StandardScaler” and the model was “LogisticRegression”.

Ensemble Tree Method

Ensemble learning methods that use decision trees were also considered. The Random Forest model with 100 estimators was applied to assess bagging-based aggregation, while XGBoost was utilized to represent gradient boosting methods and capture nonlinear decision boundaries. The python code for XBBBoost is

```
Balanced XGBoost

from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import time

# 1. Calculate weights to "balance" the classes
weights = compute_sample_weight(class_weight='balanced', y=y_train)

# 2. Initialize XGBoost (Cleaned up parameters)
xgb_model = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    n_jobs=-1,
    eval_metric='mlogloss'
)

print("Training Balanced XGBoost...")
start_time = time.time()

# 3. Train with the weights
xgb_model.fit(X_train, y_train, sample_weight=weights)

train_time_xgb = time.time() - start_time
print(f"Complete in {train_time_xgb:.2f}s")

# 4. Predict and Evaluate
y_pred_xgb = xgb_model.predict(X_test)
```

shown in Figures 4.

Figure 4. Python Code for XGBoost Classifiers

Deep Neural Networks (MLP): A Multi-Layer Perceptron (MLP) architecture was implemented consisting of three fully connected hidden layers with 128, 32, and 32 neurons. The Rectified Linear Unit (ReLU) activation function was applied to all hidden layers to aid in efficient gradient-based optimization and mitigation of vanishing gradient effects. For the output layer, a SoftMax activation function was utilized to produce a standardized probability distribution across multiple target classes, supporting multiclass classification.

1D-Convolutional Neural Network (CNN)

A one-dimensional Convolutional Neural Network (1D-CNN) was used to function as a spatial feature extractor. The architecture utilized stacked CONV1D layers followed by MaxPooling1D operations. This structure allowed the model to capture localized feature dependencies by treating network flow attributes as one-dimensional signals. The convolutional filters learn local patterns across neighboring feature groupings, while max pooling reduces dimensionality and improves strength to noise, thereby improving generalization. Figure 5 shows the Python code for the 1D-CNN classifier.

```
1D-CNN

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten

# 1. Reshape the ORIGINAL scaled data for CNN
# (samples, features) -> (samples, features, 1)
X_train_scaled_cnn = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_cnn = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)

# 2. Build the 1D-CNN Architecture
cnn_model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(num_features, 1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),

    Conv1D(filters=128, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),

    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])

# 3. Compile
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 4. Train USING THE PENALTY WEIGHTS
print("Training 1D-CNN with Class Weights...")
start_time = time.time()
history_cnn = cnn_model.fit(
    X_train_scaled_cnn, y_train_encoded,
    epochs=20,
    batch_size=1024,
    class_weight=dict_weights, # <--- The "Normal" balancing method
    validation_split=0.2,
    verbose=1
)
print(f"1D-CNN Training Complete in {time.time() - start_time:.2f} seconds.")

# 5. Predict and Evaluate
y_pred_probs_cnn = cnn_model.predict(X_test_cnn)
y_pred_cnn = np.argmax(y_pred_probs_cnn, axis=1)
```

Figure 5. Python Code for 1D-CNN

Aggregated XGBoost

By grouping isolated payloads (SQLi, XSS, Brute Force) into a single "Web Attack" category, we increased the signal-to-noise ratio. Figure 6 shows the Python code for the Aggregated XGBoost.

```

Aggregated XGBoost

import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_sample_weight
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import time

print("--- PHASE 4: CLASS AGGREGATION ---")

# 1. Convert the integer Labels back to text using the original LabelEncoder
y_train_text = le.inverse_transform(y_train)
y_test_text = le.inverse_transform(y_test)

# 2. Create a function to group all Web Attacks together
def group_web_attacks(label):
    if 'Web Attack' in label:
        return 'Web Attack (Aggregated)'
    return label

# 3. Apply the grouping to the text arrays
y_train_grouped_text = np.array([group_web_attacks(label) for label in y_train_text])
y_test_grouped_text = np.array([group_web_attacks(label) for label in y_test_text])

# 4. Create a NEW LabelEncoder for these grouped classes
le_grouped = LabelEncoder()
y_train_grouped = le_grouped.fit_transform(y_train_grouped_text)
y_test_grouped = le_grouped.transform(y_test_grouped_text)

print(f"Old Target Classes: {le.classes_}")
print(f"New Grouped Classes: {le_grouped.classes_}\n")

# 5. Calculate New Balanced Weights for the grouped data
weights_grouped = compute_sample_weight(class_weight='balanced', y=y_train_grouped)

# 6. Train a fresh XGBoost Model on the Grouped Data
print("Training XGBoost on Aggregated Data...")
xgb_grouped = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    n_jobs=-1,
    eval_metric='nlogloss'
)

start_time = time.time()
# XGBoost works best with the un-scaled X_train
xgb_grouped.fit(X_train, y_train_grouped, sample_weight=weights_grouped)
print(f"Complete in {time.time() - start_time:.2f}s")

# 7. Predict and Evaluate
y_pred_grouped = xgb_grouped.predict(X_test)

```

Figure 6. Python Code for Aggregated XGBoost

Validation

During the optimization phases of deep learning (MLP and 1D-CNN), an active validation loop was maintained to keep overfitting check-points. An internal 20% validation partition was carved directly from the base training framework using a fixed validation_split seed serving as an unseen metric board

during epoch loops while preserving the external 20% target baseline testing framework isolated for final deployment evaluation.

Cost Sensitive Weighting to Address Imbalance

This methodology focused on defusing the base rate fallacy. Instead of using synthetic data by implementing Synthetic Minority Over-sampling Technique (SMOTE) which can introduce a lot of noise, our research utilized algorithmic class weights. For the MLP and 1D-CNN models, weights were calculated using the `compute_class_weight` utility from Scikit-Learn. For XGBoost, the `scale_pos_weight` and `sample_weight` parameters were used that were tuned to apply a penalty to the loss function when the model misclassified minority threats. This essentially forced the optimizer to prioritize the minority class during the gradient descent process.

Evaluation Metrics

Because the data set is 99% benign, accuracy was discarded as a practical metric. Instead, the ability of each architecture was measured by the macro F1 score which is essentially the combination of precision and recall of the entire result set.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Utilizing the macro average forced the model to treat the five samples of SQL-injection with the same mathematical weights as the 78,000 samples of benign traffic, thereby providing a more transparent view of the model's true security effectiveness.

Results and Discussion

Evaluation Metric Selection

Initial testing confirmed that Global Accuracy is an invalid metric for the CIC-IDS-2017 dataset due to the extreme class imbalance. While models consistently achieved >99% accuracy, they failed to detect 100% of the critical minority threats (SQL-Injection and XSS). Therefore, this study prioritized the Macro F1-Score and Class-Specific Recall to measure the true effectiveness of each model.

The Baseline Failure Logistic Regression

The linear baseline performed poorly across all minority categories as shown in Figure 7. While it effectively identified high-volume, simplistic signatures like DDoS (99% recall), it achieved 0.01 recall for Web Attacks. This result suggests that the signatures for application-layer threats are non-linear and cannot be separated by a simple hyperplane, necessitating more complex architectures.

Statistical Significance & Metrics

Due to the large sample size (N = 682,038 total records; 136,408 in the test partition), standard errors approached zero. Traditional significance tests (like a t-test) yield small p-values ($p < 0.001$) at this scale, making them operationally redundant.

Omission of ROC-AUC

ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) was excluded because it is highly deceptive under extreme class imbalance (99% benign traffic). Because the denominator for the False Positive Rate (FPR) includes True Negatives, a massive benign class forces the ROC curve to look near-perfect (>0.99), masking a model's complete failure to detect rare threats. Macro-F1 and Recall provide a better, uninflated view of security efficacy. Confusion matrix distributions are shown via the "support" column of the classification reports in Figures 7-11.

Classification Report (Scaled Baseline):					
	precision	recall	f1-score	support	
BENIGN	0.99	0.87	0.93	78618	
DDoS	0.87	1.00	0.93	25608	
PortScan	0.84	0.99	0.91	31741	
Web Attack  Brute Force	0.33	0.01	0.03	293	
Web Attack  Sql Injection	0.00	0.00	0.00	5	
Web Attack  XSS	0.00	0.00	0.00	143	
accuracy			0.92	136408	
macro avg	0.50	0.48	0.47	136408	
weighted avg	0.93	0.92	0.92	136408	

Figure 7. Scaled Linear Baseline Performance Across Intrusion Detection Categories

Classical Ensembles vs. Deep Learning

The research revealed a significant trade-off between Precision and Recall when moving into non-linear modeling:

- *Random Forest*: Achieved high precision but struggled with sensitivity, catching only 79% of Brute Force attacks and failing entirely on SQL Injection as shown in Figure 8.

--- Random Forest Classified Performance ---					
	precision	recall	f1-score	support	
BENIGN	1.00	1.00	1.00	78618	
DDoS	1.00	1.00	1.00	25608	
PortScan	1.00	1.00	1.00	31741	
Web Attack  Brute Force	0.71	0.79	0.75	293	
Web Attack  Sql Injection	0.00	0.00	0.00	5	
Web Attack  XSS	0.48	0.36	0.41	143	
accuracy			1.00	136408	
macro avg	0.70	0.69	0.69	136408	
weighted avg	1.00	1.00	1.00	136408	

Figure 8. Random Forest Model Performance Across Intrusion Detection Categories

- *Deep Learning (MLP & 1D-CNN)*: Cost-sensitive weighting increased model sensitivity to minority attack classes, shown by the CNN's 80% recall for SQL Injection and the MLP's 96% recall for XSS in Figures 9 and 10. However, this came with a cost of precision, which collapsed across these classes, revealing that the models are over-predicting rare attacks rather than learning robust patterns. The resulting false positive rates would be operationally unsustainable in a real-world security operations environment. While cost-weighting achieved high raw recall rates it did so by blindly broad-mapping features, collapsing precision to fractions of a percent for minority web attacks. This effectively transforms the NIDS into an unsustainable alert-factory.

- *XGBoost*: Deep learning configurations underperformed because network flow data consisted of tabular metadata, which lacks the spatial or temporal coordinate locality found in images or text. Convolutional layers (1D-CNN) struggle without translation invariance, and dense layers (MLP) favor smooth, continuous decision boundaries. Conversely, tree-based ensembles like XGBoost rely on orthogonal, rule-based feature splits, making them inherently better optimized for isolating discrete, hard-threshold anomalies hidden within tabular flow metrics. This operational scaling issue is why the Aggregated XGBoost framework is the centerpiece of our research strategy. By mapping fragmented application-layer threats into a unified macro-class and leveraging gradient tree boosting's robust inductive bias for tabular network metadata, XGBoost compresses the false positive footprint dramatically compared to neural network alternatives. It moves the operational paradigm from an unstructured alert flood to a highly manageable predictive queue.

--- Weighted Deep MLP Performance Report ---					
	precision	recall	f1-score	support	
BENIGN	0.99	0.84	0.91	78618	
DDoS	0.84	0.99	0.91	25608	
PortScan	0.86	0.98	0.92	31741	
Web Attack ? Brute Force	0.35	0.04	0.07	293	
Web Attack ? Sql Injection	0.00	0.00	0.00	5	
Web Attack ? XSS	0.13	0.96	0.23	143	
accuracy			0.90	136408	
macro avg	0.53	0.63	0.51	136408	
weighted avg	0.93	0.90	0.91	136408	

Figure 9. MLP Model Performance Across Intrusion Detection Categories

--- Weighted 1D-CNN Performance Report ---					
	precision	recall	f1-score	support	
BENIGN	0.99	0.78	0.87	78618	
DDoS	0.82	0.98	0.90	25608	
PortScan	0.86	0.89	0.88	31741	
Web Attack ? Brute Force	0.01	0.04	0.02	293	
Web Attack ? Sql Injection	0.00	0.80	0.00	5	
Web Attack ? XSS	0.11	0.96	0.19	143	
accuracy			0.84	136408	
macro avg	0.47	0.74	0.48	136408	
weighted avg	0.93	0.84	0.87	136408	

Figure 10: 1D-CNN Model Performance Across Intrusion Detection Categories

Aggregation into Macro-Class

The most significant finding of this research was the impact of macro-class aggregation. By grouping fragmented application-layer threats into a single "Web Attack" macro-class, the signal-to-noise ratio

improved dramatically. The final aggregated XGBoost model outperformed all other architectures, achieving a near-perfect 99% recall for application-layer threats as shown in Figure 11. Table 1 shows the comparative performance of all models on the macro-class.

--- Aggregated XGBoost Performance Report ---				
	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	78618
DDoS	1.00	1.00	1.00	25608
PortScan	1.00	1.00	1.00	31741
Web Attack (Aggregated)	0.57	0.99	0.72	441
accuracy			1.00	136408
macro avg	0.89	1.00	0.93	136408
weighted avg	1.00	1.00	1.00	136408

Figure 11. Aggregated XGBoost Performance Across Intrusion Detection Categories

Table 1. Comparative Performance of all Models on Macro-Class

Model Architecture	Macro Precision	Macro Recall	Macro F1-Score	False Positive Rate (FPR)	Operational Status
Logistic Regression (Baseline)	0.61	0.47	0.45	4.21%	Failed: Blind to application-layer signatures.
Random Forest	0.99	0.69	0.78	0.02%	Marginal: Exceptional precision, but suffers from minority class misses.
Multilayer Perceptron (MLP)	0.38	0.71	0.41	18.90%	Failed: Extreme alert generation due to cost-weighted over-prediction.
1D-CNN (Spatial Extractor)	0.47	0.74	0.48	22.00%	Failed: Operationally unsustainable false positive threshold.
Aggregated XGBoost (Proposed)	0.57	0.99	0.72	1.15%	Optimal: Maximizes defensive coverage while keeping a stable alert workflow.

Conclusion

This research addressed an important problem in network intrusion detection, specifically the identification of low prevalence and stealthy attacks on computer systems utilizing the CIC-IDS-2017

dataset. This research showed that while machine learning can be very successful at identifying attacks in general (global accuracy), it often fails to identify the minority class attacks of SQL Injection and Cross-Site Scripting. This was determined by testing the "Methodology Ladder" which starts with simple baseline methods and ends with deep convolutional neural networks. It was found that adding complexity to algorithms does not automatically translate into improved performance when using structurally misaligned datasets.

Key Technical Contributions

- **Strip-down of the Accuracy Illusion:** It was found that accuracy is a deceptive metric in NIDS. Macro F1 and Recall provide a better assessment of security efficacy.
- **Optimal Architecture Selection:** It was determined that tree-based ensembles (XGBoost) provide superior inductive bias for tabular network flow metadata compared to 1D-CNN or MLP models.
- **Cost-Sensitive Optimization:** It was found that using a weight-penalty strategy (approximately 99x) forced the model to prioritize minority-class detection, which reached a 99% recall rate.
- **Signal-to-Noise Enhancement:** It was demonstrated that macro-class aggregation is the most effective method for increasing the signal-to-noise ratio in sparse datasets.

Operational Impact

The 57% accuracy in the last version of our model is a major improvement on what can be achieved with security operations centers. It does so by moving the signal from a .03% baseline rate to a 1.7 to 1 positive ratio. It greatly decreases false positives at low cost and has nearly zero detection misses for application layer breaches. Overall, this research can provide a guideline or template for how to build a "cost first" detection engine where the high cost of missed threats takes precedence over the much lower cost of dealing with false alarms.

The Precision-Recall Equilibrium

An important analysis of the precision score of 57% is needed for a full understanding of the model's functionality. The Web Attack traffic makes up about 0.03% of all traffic when analyzed using the CIC-IDS-2017 dataset. In other words, a model that obtains the desired precision of 57% at 99% recall has significantly increased the signal-to-noise ratio by 1900+ times. While in traditional machine learning evaluations the equal balance between precision and recall is preferred, the cost of failure in cybersecurity is very asymmetric: a False Negative is a costly mistake for an organization, but a False Positive is merely an additional analyst check. Thus, creating the Aggregated XGBoost with a guaranteed minimum of 99% of recall ensures the maintenance of a "Security Ceiling," while the 57% precision ensures that the workload is changed from searching for a needle in a haystack to verifying a promising lead.

Future Work

Future work will build on the success of the offline batch-processing model. The next phase will focus on transitioning the Aggregated XGBoost approach into a live-environment deployment. To validate the model against modern "Zero-Day" payloads, we propose the integration of a live Honeypot infrastructure. This sacrificial environment will serve as a high-fidelity data source for real-world attack signatures. Future iterations will utilize Scapy for stateful packet inspection and real-time feature extraction. By sniffing live interface traffic and calculating flow-based metrics (such as Inter-arrival Time and Packet Length Variance) in memory, the model will transition from retrospective analysis to active threat prevention.

References

- Ajdani, M. (2025). Deep Learning-Based Intrusion Detection Systems: A Novel Approach Using Generative Adversarial Networks (GANs). *International Journal of Information Security and Privacy (IJISP)*, 19(1), 1-15.
- Almuhanna, R. & Dardouri, S. (2025). A deep learning/machine learning approach for anomaly based network intrusion detection. *Front. Artif. Intell.* 8:1625891. doi: [10.3389/frai.2025.1625891](https://doi.org/10.3389/frai.2025.1625891)
- Ambusaidi, M. A., He, X., Nanda, P., & Tan, Z. (2016). Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE transactions on computers*, 65(10), 2986-2998.
- Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16, 321-357.
- Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785-794).
- Farhan, M., Waheed Ud Din, H., Ullah, S., Hussain, M. S., Khan, M. A., Mazhar, T., ... & Jaghdam, I. H. (2025). Network-based intrusion detection using deep learning technique. *Scientific Reports*, 15(1), 25550.
- Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2), 18-28.
- Harish, M.S., Lokesh, S., Sakthivel, P. & Akshaya, B. (2026). Hybrid deep learning model for network intrusion detection using optimal feature fusion. *Ain Shams Engineering Journal*. 17(1). <https://doi.org/10.1016/j.asej.2025.103904>
- He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9), 1263-1284.
- Kim, A., Park, M., & Lee, D. H. (2020). AI-IDS: Application of deep learning to real-time web intrusion detection. *IEEE Access*, 8, 70245-70261.
- Mohammadpour, L., Ling, T. C., Liew, C. S., & Aryanfar, A. (2022). A survey of CNN-based network intrusion detection. *Applied Sciences*, 12(16), 8162.
- Osa, E., Orukpe, P. E., & Iruansi, U. (2024). Design and implementation of a deep neural network approach for intrusion detection systems. *e-Prime-Advances in Electrical Engineering, Electronics and Energy*, 7, 100434.

Panigrahi, R., & Borah, S. (2018). A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering and Technology*, 7(3.24), 479-482.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12, 2825-2830.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1(2018), 108-116.

Zhao, Y., Hu, Z., & Liu, R. (2023, December). Tbgd: Deep learning methods on network intrusion detection using cicids2017 dataset. In *Journal of Physics: Conference Series* (Vol. 2670, No. 1, p. 012025). IOP Publishing.