Golden Sun-Rise
International Journal of Multidisciplinary on Science and Management
ISSN: 3048-5037/ Volume 2 Issue 2 Apr-Jun 2025 / Page No: 01-16
Paper Id: IJMSM- V2I2P101/ Doi:10.71141/30485037/V2I2P101

Research Article

Detecting Unbalanced Network Traffic : A Machine Learning Using Stacked Generalization

Vitthal B. Kamble[1], Kunal R. Jadhav[2], Tanesh M. Patil[3], Ayush D. More[4]

[1]*Computer Engineering, Cusrow Wadia Institute of Technology Pune, Maharastra, India*.

Received: 20 March 2025 Revised: 25 March 2025 Accepted: 28 March 2025 Published: 05 April 2025

Abstract - *Cyber Threats are becoming more frequent and sophisticated, so we need better systems to detect malicious activities in network traffic. Detecting unbalanced network traffic is a critical challenge in cyber security, where malicious activities are often underrepresented in comparison to legitimate traffic. This study proposes a hybrid approach using XG- Boost, Random Forest, and an ensemble model to effectively identify anomalies in network traffic data. We also use dataset of IDS. Our approach improves the accuracy, efficiency, and reliability of intrusion detection, contributing to stronger defenses against cyber attacks and protecting important network systems*

Keywords - *Cyber security, Intrusion Detection System (IDS), XG-Boost, Random Forest, Ensemble Model, Anomaly Detection, Class Imbalance, Network Security, Machine Learning, Hybrid Model, Threat Detection, Data Preprocessing, Malicious Traffic Detection.*

I. INTRODUCTION

In today's interconnected world, where digital communication is the backbone of many essential systems and services, maintaining network security has become a top priority. Cyber attacks are becoming more sophisticated, frequent, and challenging for organizations that need to protect sensitive data, ensure business continuity, and maintain public trust. Among various cyber threats, network intrusions pose a significant risk. These intrusions involve unauthorized access to systems, data manipulation, or network disruption, which can lead to severe financial and operational losses.

A. Why Traditional Methods Fail

Traditional intrusion detection systems (IDS) often rely on rule-based methods, signature detection, or anomaly-based approaches[21]. Rule- based detection uses pre-defined patterns of known attacks, but these systems struggle to detect new or evolving threats. Anomaly detection can be more flexible, identifying deviations from normal network behavior. However, these methods are prone to false positives, where legitimate activities are incorrectly flagged as malicious. Moreover, sophisticated cyber attacks often blend in with regular traffic, making them harder to detect with conventional techniques.

B. Challenges with Unbalanced Network Traffic

A critical issue in network traffic analysis is the imbalance between malicious and legitimate traffic. In most networks, the vast majority of data consists of normal user activities, while malicious attempts represent a tiny fraction. This creates a highly skewed dataset where conventional machine learning models often fail. They tend to be biased toward the majority class, neglecting minority class instances which are often the cyber attacks we need to detect. To illustrate, imagine a multi-lane highway with heavy traffic in some lanes and minimal activity in others. Attackers may exploit the less-monitored lanes to execute their malicious activities undetected. Similarly, within networks, cybercriminals may target underused systems or disguise attacks as legitimate activities to bypass detection system.

C. The Role of Machine Learning in Intrusion Detection

Machine learning (ML) offers a robust solution by analyzing large amounts of network data, learning patterns, and detecting anomalies in real time. Unlike static, rule-based systems, ML models can continuously adapt to new and emerging threats. By using algorithms like XG-Boost and Random Forest, we can enhance the accuracy and reliability of intrusion detection.

- **XG-Boost (Extreme Gradient Boosting)**: XG-Boost is an efficient and powerful ML algorithm widely used for structured data analysis. It uses a boosting approach, where multiple decision trees are combined to reduce errors and improve predictive accuracy. Its ability to handle missing data, prevent over fitting, and optimize computational performance makes it ideal for network traffic classification.
- **Random Forest**: Random Forest is a versatile and robust ensemble learning algorithm that uses multiple decision trees to improve classification performance. By averaging the predictions from multiple trees, it reduces over fitting and enhances model generalization. Random Forest is particularly effective when dealing with large datasets and complex patterns in network traffic.
- **Ensemble Model**: To further improve performance, we can combine the strengths of both XG-Boost and Random Forest using a stacking ensemble model. In this approach, the predictions from both models are used as input features for a secondary model, often a logistic regression classifier, which makes the final prediction. This combination often results in higher accuracy, improved generalization, and better detection of malicious activities.

II. RELATED WORKS

A number of papers deals with top points related to various aspects of cloud computing, cyber security and ethical hacking[1].They investigate frameworks for selecting optimal cloud services, predicting service rankings, and addressing challenges in cloud-based software development [2]. Additionally, performance analysis of encryption algorithms in cloud computing is examined [3]. These studies contribute to understanding cloud computing's ef5iciency and security. Furthermore, they highlight the growing importance of cyber security, as indicated by market forecasts predicting signi5icant growth in the cyber security industry [4]. Overall, the research provides valuable insights into improving cloud service selection, predicting service rankings, addressing development challenges, and enhancing cyber security measures in the digital era [5]. This study presents a novel dimensionality reduction strategy for detecting Distributed Denial of Service (DDoS) attacks in cloud computing environments [6].

Focusing on the autonomous detection of malicious events using machine learning models in drone networks [7], it also introduces a machine-learning-enabled intrusion detection system for cellular- connected UAV networks [8]. Additionally, this study presents a lightweight IDS for UAV networks utilizing a periodic deep reinforcement learning- based approach [9]. Furthermore, arti5icial intelligence is leveraged for intrusion detection systems in unmanned aerial vehicles [10], while a high-performance intrusion detection system for networked UAVs is developed using deep learning techniques [11]. Additionally, a data normalization technique is proposed for detecting cyber- attacks on UAVs [12].Another method involves creating a system with multiple layers of security using a mix of technologies called a hybrid Deep Belief Network [13].Researchers also explored using advanced algorithms like Particle Swarm Optimization along with Deep Belief Networks to improve the accuracy of intrusion detection systems [14]. Additionally, the study discusses using bio-inspired models and hybrid deep learning techniques to make network security more robust [15]. Ethical hacking has emerged as a crucial practice, enabling organizations to fortify their defenses and safeguard their assets[17].Detect UPI Fraud By Using Machine learning[41].

III. METHODOLOGY

A. Data Collection

The CIC-IDS2017 (Canadian Institute for Cyber security - Intrusion Detection System 2017) dataset is a comprehensive and realistic network traffic dataset designed for cyber security research[20]. It was created to simulate real-world network traffic, including both normal activities and various types of cyber attacks. Researchers and cyber security experts use this dataset to evaluate intrusion detection systems (IDS) and machine learning models. Purpose of CIC-IDS 2017:

- Designed to address the challenges in detecting modern cyber attacks.
- Provides labeled network traffic data for supervised learning.
- Enables the development and evaluation of machine learning-based intrusion detection systems.

- Contains diverse attack scenarios to test model robustness.

Attacks In CIC-IDS2017: The dataset covers 15 distinct types of attacks.

1. Brute Force Attacks
    - SSH-Brute Force
    - FTP-Brute Force
2. Denial of Service (DoS)
    - DoS-Slowloris
    - DoS-GoldenEye
    - DoS-Hulk
    - DoS-SlowHTTPTest
3. Distributed Denial of Service(DDoS)
    - DDoS Attack
4. Web-Based Attacks
    - Web Shell, SQL Injection
    - XSS(Cross-Site Scripting)
5. Infiltration
6. Port Scanning
7. Botnet Activity
8. Data Exfiltration

B. Data Preprocessing

Effective data preprocessing is a crucial step in ensuring the accuracy and efficiency of machine learning models[18]. The following preprocessing techniques were applied to the CIC-IDS2017 dataset to clean and prepare the data for model training:

a. Data Cleaning

Data cleaning was performed to handle missing, inconsistent, and infinite values. Missing data can lead to biased models and inaccurate predictions. Rows containing missing values were removed, and infinite values (resulting from computational errors) were replaced with Nan and subsequently dropped. This step ensures a clean and reliable dataset for model training.

b. Label Encoding

The dataset contains categorical features such as protocol types and attack labels. These categorical variables were converted to numerical representations using Label Encoding. Each unique category was assigned a unique integer value. Label Encoding helps machine learning models interpret categorical data more efficiently by converting it into a numerical format.

c. Feature Scaling

To ensure uniformity and optimize model performance, Standard Scaling was applied to normalize the feature values. Standard Scaling transforms the data such that it has a mean of zero and a standard deviation of one. This is particularly beneficial for models like XG-Boost and Random Forest, where variations in feature scales may otherwise impact performance.

d. Feature Selection using Variance Thresholding

Variance Thresholding was employed to remove low-variance features that provide little to no useful information. Features with near-constant values were eliminated as they contribute minimal predictive power. By applying a threshold of 0.01, only the most relevant features with sufficient variance were retained for model training.

The Datasets is then split into training and testing sets using an 80-20 split. These preprocessing steps collectively enhance model accuracy, reduce computational complexity, and improve the generalizability of the models. The cleaned and processed dataset was then used for training and testing , evaluating machine learning models, including XG-Boost, Random Forest, and an ensemble of both.

C. Handling Unbalanced Data

In imbalanced datasets like CIC-IDS2017, where malicious network traffic is significantly underrepresented compared to legitimate traffic, traditional machine learning models often struggle to correctly identify minority class instances. This leads to biased predictions, poor recall for minority classes, and a lower overall detection accuracy. SMOTE is an effective solution to address this issue by generating synthetic samples for the minority classes rather than duplicating them. It creates new data points by interpolating between existing minority class instances, making the dataset more balanced. This synthetic generation helps the model learn more effectively from the minority class patterns, improving its generalization and reducing the risk of over fitting.

a. Why SMOTE?

- **Improves Model Performance**: Enhances the classifier's ability to detect minority class instances, leading to better recall and F1-score.
- **Reduces Class Imbalance**: Generates realistic synthetic data that balances the dataset without introducing redundancy.
- **Minimizes Overfitting**: Unlike random oversampling, SMOTE prevents the model from memorizing repeated data points by creating diverse synthetic samples.
- **Efficient for Large Datasets**: SMOTE is computationally efficient and scalable for large datasets like CIC- IDS2017.
- **Sampling Strategy and Parameter Tuning**:The effectiveness of SMOTE depends on the choice of parameters and the sampling strategy.

In this study, the following strategies were used:

1. **Sampling Strategy**
   - Sampling_strategy='auto': SMOTE was applied to balance the classes by increasing the number of minority samples to match the majority class.
   - For datasets with multiple classes, a customized strategy could be applied to balance only the severely underrepresented classes.
2. **K-Nearest Neighbors (K_neighbors)**
   - K_neighbors=5: SMOTE uses a k-nearest neighbors approach to generate synthetic samples by selecting five nearest neighbors of a given minority class instance.
   - This parameter was tuned by evaluating model performance across different k values. Lower k values reduce the risk of introducing noise, while higher values ensure smoother interpolation.
3. **Random State**
   - Random_state=42: A fixed random state ensured reproducibility of the results across multiple runs.
4. **Edge Cases Management**
   - In cases where the number of minority samples was extremely low (less than 5 samples), the k_neighborsparameter was dynamically adjusted using:

     k_neighbors = min(5, minority_class_count - 1) if minority_class_count > 1 else

D. Model Implementation

a. XG-Boost Classifier

XG-Boost (Extreme Gradient Boosting) is a highly efficient and scalable machine learning algorithm based on the gradient boosting framework[19]. It is designed to improve prediction accuracy by combining the outputs of multiple weak learners,typically decision trees. XG-Boost is particularly effective for handling structured tabular data and is widely used in classification tasks, including those involving imbalanced data.

- **Gradient Boosting**: XG-Boost minimizes the loss function using a gradient descent approach, sequentially building decision trees to correct the errors of the previous models.
- **Handling Imbalanced Data**: By incorporating objective functions like logistic loss and using weighted loss functions, XG-Boost can assign higher importance to minority classes, improving recall and precision for rare events.
- **Regularization**: XG-Boost uses L1 and L2 regularization to reduce over fitting, making it robust for noisy network traffic data.

- **Feature Importance**: XG-Boost provides insights into feature importance, allowing for a better understanding of influential network attributes in anomaly detection

Equation

Accuracy=TP+TN/TP+TN+FP+FN (1)

Where:

- **TP** = True Positives (Correctly predicted positive instances)
- **TN** = True Negatives (Correctly predicted negative instances)
- **FP** = False Positives (Incorrectly predicted positive instances)
- **FN** = False Negatives (Incorrectly predicted negative instances)

Precision: Precision=TP/TP+ FP (2)

Recall=TP/TP+ FN (3)

F1Score:F1Score=2(Precision∗ Recall)/Precision+ Recall (4)

Recall (also known as Sensitivity or True Positive Rate)

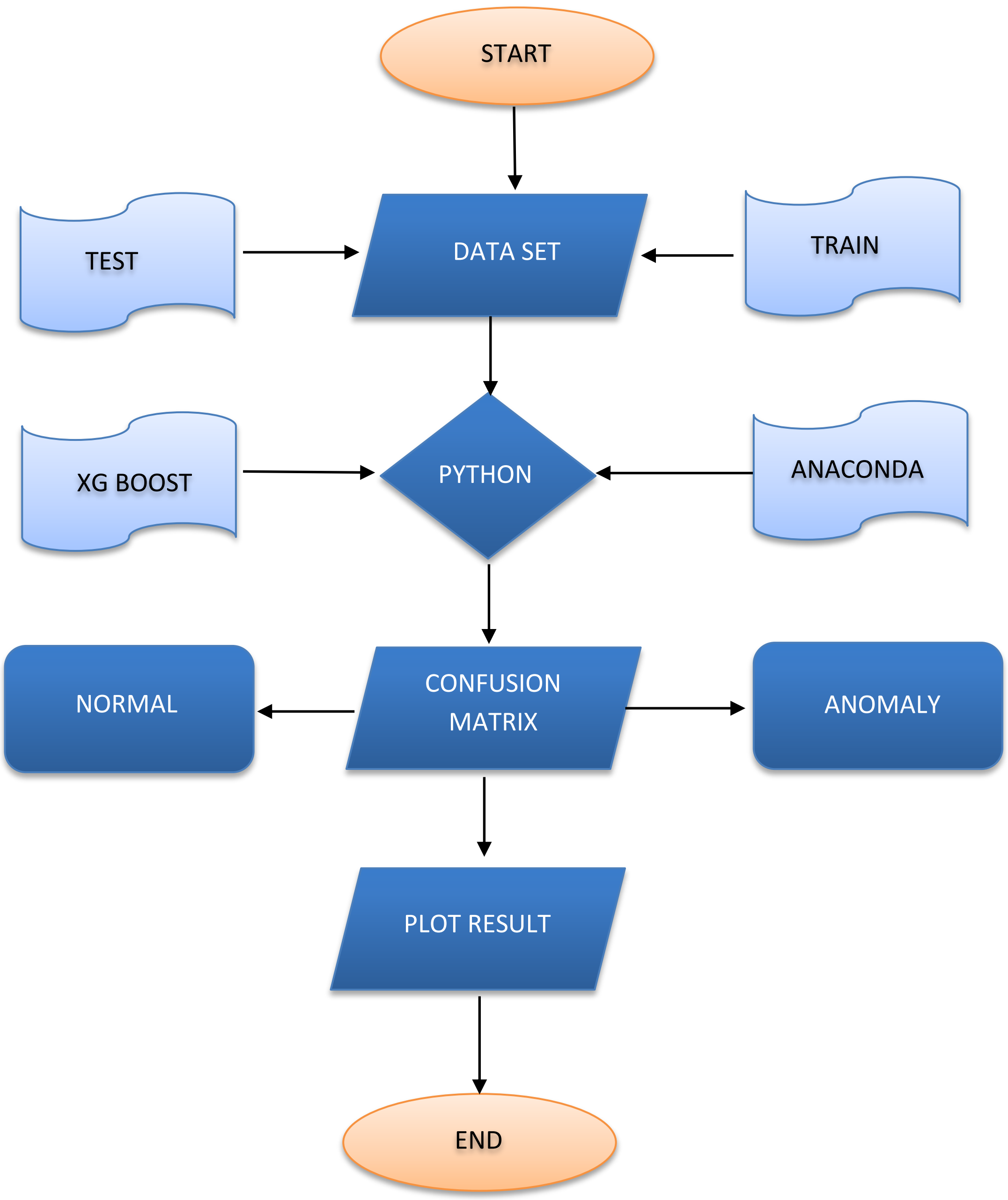


Figure 1. Flow of Work Chart

b. Random Forest Classifier

Random Forest is a powerful ensemble learning algorithm that constructs multiple decision trees using randomly selected subsets of the training data. Its predictions are made by averaging the outputs of individual trees for regression tasks or by majority voting for classification tasks.

- **Robust Predictions**: Random Forest is resilient to noise and over fitting due to its bagging technique, making it ideal for large and complex datasets.
- **Feature Importance Analysis**: t provides feature importance scores using techniques like Gini Importance or Mean Decrease in Impurity, enabling effective feature selection.
- **Handling Imbalanced Data**: By adjusting class weights and using bootstrap sampling, Random Forest can manage class imbalance effectively. It also offers stable performance in scenarios where class distributions are uneven.

Equation

Accuracy=TP+TN/TP+TN+FP+FN (5)

Where:

- TP = True Positives(Correctly predicted positive instances)
- TN = True Negatives(Correctly predicted negative instances)
- FP = False Positives(Incorrectly predicted positive instances)
- FN = False Negatives(Incorrectly predicted negative instances)

Precision: Precision=TP/TP+ FP (6)

Recall= TP / TP+ FN (7)

F1Score: F1Score=2(Precision$*$ Recall)/Precision+ Recall (8)

Balanced Accuracy: Balanced Accuracy = Sensitivity + Specificity/2 (9)

Recall (also known as Sensitivity or True Positive Rate)

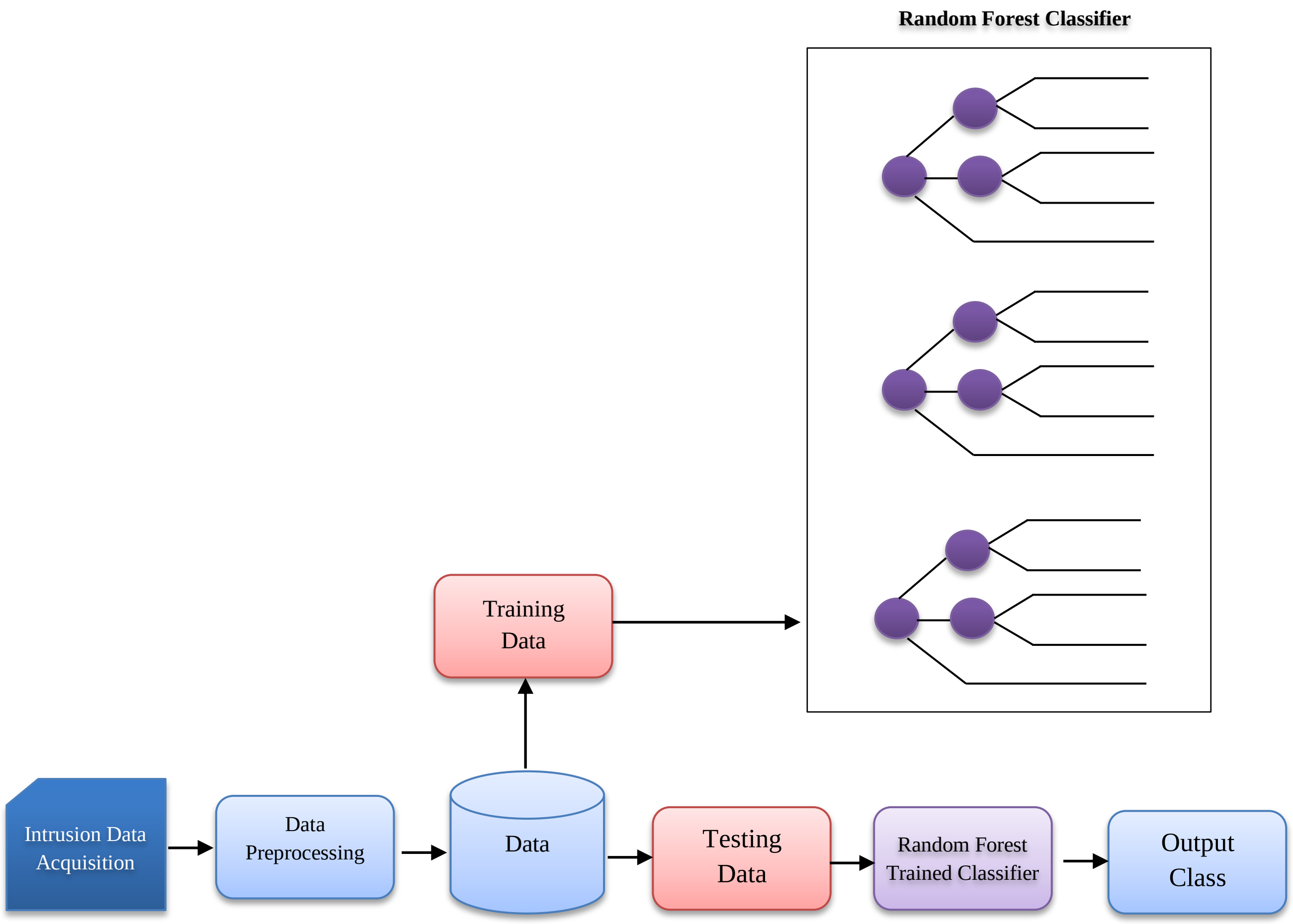


Figure 2. Virtual Implementation of Random Forest

c. Ensemble Model with Stacked Generalization

Instead of using Logistic Regression as a meta-classifier, stacked generalization is employed to combine the strengths of the XG-Boost and Random Forest classifiers. In this approach, the predictions from the base classifiers are treated as inputs to a secondary model, often called the meta-learner. The meta-learner is trained to learn the best way to combine these predictions to make the final decision.

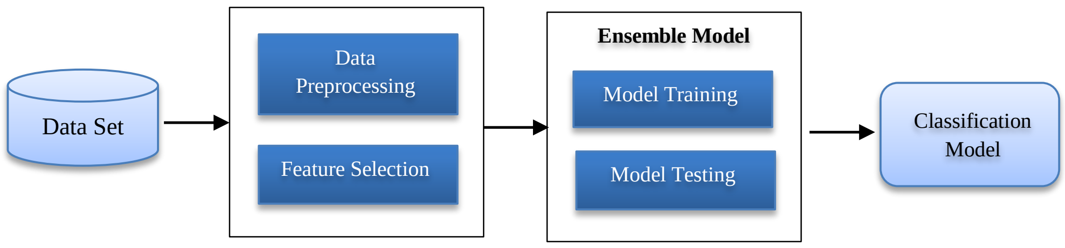


Figure 3. Creation of Ensemble Model

- **Improved Accuracy**: Stacked generalization reduces the risk of over fitting by leveraging the strengths of diverse models.
- **Robustness**: By capturing complementary information from the base models, it enhances prediction accuracy, especially for complex network traffic data.
- **Adaptive Learning**: The meta-learner adapts to the misclassifications of the base models, improving overall model reliability.
- **Implementation Flexibility**: Various machine learning algorithms, including gradient boosting models, support vector machines, or deep neural networks, can be used as meta- learners depending on the dataset and use case.

Equation

Accuracy = TP+TN/TP+TN+FP+FN (10)

Where:

- TP = True Positives(Correctly predicted positive instances)
- TN = True Negatives(Correctly predicted negative instances)
- FP = False Positives(Incorrectly predicted positive instances)
- FN = False Negatives(Incorrectly predicted negative instances)

Precision: Precision = TP/TP+ FP (11)

Recall = TP /TP+ FN (12)

F1Score: F1Score = 2(Precision∗ Recall)/Precision+ Recall (13)

Balanced Accuracy: Balanced Accuracy = Sensitivity + Specificity/2 (14)

Recall(also known as Sensitivity or True Positive Rate)

IV. EXISTINGIMPLEMENTATIONMETHODS

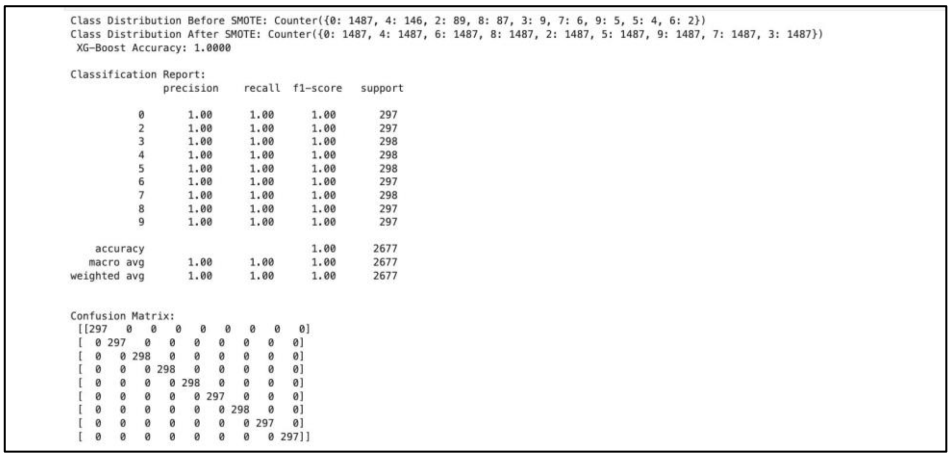
A. XG-BOOST

Implementation Code

```
#XG-Boost Algorithm
import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split from xgboost import XGBClassiCier
from sklearn.metrics import accuracy_score, classiCication_report, confusion_matrix from sklearn.preprocessing
import LabelEncoder, StandardScaler
from sklearn.feature_selection import VarianceThreshold from imblearn.over_sampling import SMOTE

# Load the dataset try:
df = pd.read_csv('netwok_trafCic.csv') except FileNotFoundError:
raise FileNotFoundError("File 'network_trafCic.csv' not found. Please check the path.")
```

```
# Data Cleaning: Handle missing values df.columns = df.columns.str.strip() df.dropna(inplace=True)

# Encode categorical variables using LabelEncoder label_encoder = LabelEncoder()
for col in df.select_dtypes(include='object').columns:
df[col] = label_encoder.Cit_transform(df[col])

# Identify the target variable target_column = 'Label'
if target_column not in df.columns:
raise KeyError("Label column not found. Available columns: " + str(df.columns))

# Separate features and target
X = df.drop(columns=[target_column]) y = df[target_column]

# Handle inCinite values
X.replace([np.inf, -np.inf], np.nan, inplace=True) X.Cillna(X.max().max(), inplace=True)

# Remove low variance features
selector = VarianceThreshold(threshold=0.01) X = pd.DataFrame(selector.Cit_transform(X))

# Normalize features using StandardScaler scaler = StandardScaler()
X = pd.DataFrame(scaler.Cit_transform(X))

# Handle imbalanced data using SMOTE smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.Cit_resample(X, y)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Initialize and train XGBoost ClassiCier
xgb = XGBClassiCier(n_estimators=200, learning_rate=0.05, max_depth=7, random_state=42) xgb.Cit(X_train,
y_train)

# Make predictions
y_pred = xgb.predict(X_test)

# Evaluate the model
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassiCication Report:\n", classiCication_report(y_test, y_pred, zero_division=1)) print("\nConfusion
Matrix:\n", confusion_matrix(y_test, y_pred))
```

Output



```
Class Distribution Before SMOTE: Counter({0: 1487, 4: 146, 2: 89, 8: 87, 3: 9, 7: 6, 9: 5, 5: 4, 6: 2})
Class Distribution After SMOTE: Counter({0: 1487, 4: 1487, 6: 1487, 8: 1487, 2: 1487, 5: 1487, 9: 1487, 7: 1487, 3: 1487})
 XG-Boost Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       297
           2       1.00      1.00      1.00       297
           3       1.00      1.00      1.00       298
           4       1.00      1.00      1.00       298
           5       1.00      1.00      1.00       298
           6       1.00      1.00      1.00       297
           7       1.00      1.00      1.00       298
           8       1.00      1.00      1.00       297
           9       1.00      1.00      1.00       297

    accuracy                           1.00      2677
   macro avg       1.00      1.00      1.00      2677
weighted avg       1.00      1.00      1.00      2677


Confusion Matrix:
 [[297   0   0   0   0   0   0   0   0]
 [  0 297   0   0   0   0   0   0   0]
 [  0   0 298   0   0   0   0   0   0]
 [  0   0   0 298   0   0   0   0   0]
 [  0   0   0   0 298   0   0   0   0]
 [  0   0   0   0   0 297   0   0   0]
 [  0   0   0   0   0   0 298   0   0]
 [  0   0   0   0   0   0   0 297   0]
 [  0   0   0   0   0   0   0   0 297]]
```

Figure 4. XGBoost Model Performance on Balanced Data (SMOTE Applied)

B. Random Forest

Implementation Code

```
#Random Forest Algorithm
import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split from sklearn.ensemble import RandomForestClassiCier
from sklearn.metrics import classiCication_report, confusion_matrix, accuracy_score from
imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Load dataset
df = pd.read_csv("netwok_trafCic.csv") # Ensure the CSV Cile is in the same directory

# Trim spaces from column names to avoid key errors df.columns = df.columns.str.strip()

# Drop missing values df.dropna(inplace=True)

# Identify the target column
target_column = "Label" if "Label" in df.columns else " Label"
# Convert categorical columns to numeric using Label Encoding label_encoders = {} for col in
df.select_dtypes(include=['object']).columns: if col
!= target_column:
le = LabelEncoder()
df[col] = le.Cit_transform(df[col]) label_encoders[col]
= le

# Split features and target variable
X = df.drop(columns=[target_column]) y
= df[target_column]

# Encode the target variable
y = LabelEncoder().Cit_transform(y)

# Print class distribution before balancing print("Class Distribution Before SMOTE:", Counter(y))

# Split dataset into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


# Convert to NumPy array for faster processing X_train = X_train.to_numpy()
X_test = X_test.to_numpy()

# Handle inCinite values & NaN
X_train = np.where(np.isinf(X_train), np.nan, X_train) X_test = np.where(np.isinf(X_test), np.nan, X_test)

# Replace NaN with max Cinite value
Cinite_max = np.nanmax(X_train[np.isCinite(X_train)]) # Get max Cinite value

X_train = np.nan_to_num(X_train, nan=Cinite_max) X_test = np.nan_to_num(X_test, nan=Cinite_max)

# Normalize features scaler
= StandardScaler()
X_train = scaler.Cit_transform(X_train) X_test = scaler.transform(X_test)

# Apply SMOTE to only small classes (avoiding memory overload) min_class_size = 100000
class_counts = Counter(y_train)
```

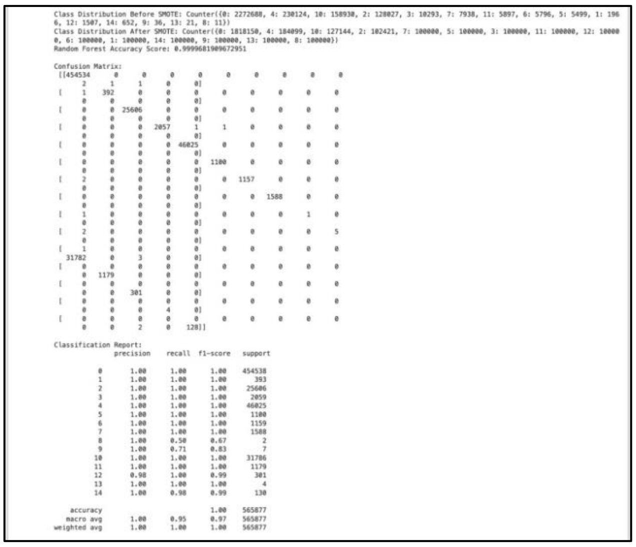
smote_dict = {cls: min_class_size for cls, count in class_counts.items() if count < min_class_size} smote = SMOTE(sampling_strategy=smote_dict, random_state=42)
X_train_resampled, y_train_resampled = smote.Cit_resample(X_train, y_train) print("Class Distribution After SMOTE:", Counter(y_train_resampled))

# Train Random Forest ClassiCier
rf_model = RandomForestClassiCier(n_estimators=100, random_state=42, class_weight="balanced", n_jobs=-1) rf_model.Cit(X_train_resampled, y_train_resampled)

# Predict on test data
y_pred = rf_model.predict(X_test)

# Evaluate model performance
print("Accuracy Score:", accuracy_score(y_test, y_pred)) print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred)) print("\nClassiCication Report:\n", classiCication_report(y_test, y_pred))

Output



```
Class Distribution Before SMOTE: Counter({0: 2272688, 4: 230124, 10: 158930, 2: 128027, 3: 10293, 7: 7938, 11: 5897, 6: 5796, 5: 5499, 1: 196
6, 12: 1507, 14: 652, 9: 36, 13: 21, 8: 11})
Class Distribution After SMOTE: Counter({0: 1818150, 4: 184099, 10: 127144, 2: 102421, 7: 100000, 5: 100000, 3: 100000, 11: 100000, 12: 10000
0, 6: 100000, 1: 100000, 14: 100000, 9: 100000, 13: 100000, 8: 100000})
Random Forest Accuracy Score: 0.9999681909672951

Confusion Matrix:
 [[454534      0      0      0      0      0      0      0      0      0
       2      1      1      0      0]
 [     1    392      0      0      0      0      0      0      0      0
       0      0      0      0      0]
 [     0      0  25606      0      0      0      0      0      0      0
       0      0      0      0      0]
 [     0      0      0   2057      1      1      0      0      0      0
       0      0      0      0      0]
 [     0      0      0      0  46025      0      0      0      0      0
       0      0      0      0      0]
 [     0      0      0      0      0   1100      0      0      0      0
       0      0      0      0      0]
 [     2      0      0      0      0      0   1157      0      0      0
       0      0      0      0      0]
 [     0      0      0      0      0      0      0   1588      0      0
       0      0      0      0      0]
 [     1      0      0      0      0      0      0      0      1      0
       0      0      0      0      0]
 [     2      0      0      0      0      0      0      0      0      5
       0      0      0      0      0]
 [     1      0      0      0      0      0      0      0      0      0
   31782      0      3      0      0]
 [     0      0      0      0      0      0      0      0      0      0
       0   1179      0      0      0]
 [     0      0      0      0      0      0      0      0      0      0
       0      0    301      0      0]
 [     0      0      0      0      0      0      0      0      0      0
       0      0      0      4      0]
 [     0      0      0      0      0      0      0      0      0      0
       0      0      2      0    128]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    454538
           1       1.00      1.00      1.00       393
           2       1.00      1.00      1.00     25606
           3       1.00      1.00      1.00      2059
           4       1.00      1.00      1.00     46025
           5       1.00      1.00      1.00      1100
           6       1.00      1.00      1.00      1159
           7       1.00      1.00      1.00      1588
           8       1.00      0.50      0.67         2
           9       1.00      0.71      0.83         7
          10       1.00      1.00      1.00     31786
          11       1.00      1.00      1.00      1179
          12       0.98      1.00      0.99       301
          13       1.00      1.00      1.00         4
          14       1.00      0.98      0.99       130

    accuracy                           1.00    565877
   macro avg       1.00      0.95      0.97    565877
weighted avg       1.00      1.00      1.00    565877
```

Figure 5. Random Forest Model Performance on Large-Scale Data (SMOTE Applied)

V. IMPLEMENTATION OF HYBRID TECHNIQUE

Combining XG-Boost and Random Forest is a powerful strategy for leveraging the strengths of both models. This hybrid approach can lead to improved performance, as it combines the robustness of Random Forest's bagging with the accuracy-enhancing properties of XG-Boost boosting.

A. Combination of XG-Boost and Random Forest

There are several ways to combine XG-Boost and RandomForest to enhance their performance:

a. Stacking (Model Stacking)

Stacking is an ensemble technique where multiple models are trained, and a meta-model is used to combine their predictions. The idea is to stack XG-Boost and RandomForest as base models and use another model (often a simpler model like logistic regression or another tree-based method) to combine the predictions of the base models.

i). Process

- Train RandomForest and XG-Boost independently on the training data.
- Make predictions from each base model.
- Feed the predictions of both models as input into a meta-model, which could be a simple linear regression or logistic regression.

ii). Advantages

- Combines the strengths of both models: the randomness and generalization of Random Forest and the fine-tuned accuracy of XG-Boost.
- Can significantly improve model performance if both models have complementary strengths.

iii). Example

- Use Random Forest to capture general patterns.
- Use XG-Boost to focus on correcting the errors made by Random Forest.
- A meta-model (e.g., logistic regression) can combine these predictions for a more accurate result.

B. Bagging + Boosting

In this approach, you can combine the bagging mechanism of Random Forest with the boosting mechanism of XG-Boost in a hybrid model.

i). Process

- Train Random Forest (bagging) on the training data.
- Use the predictions of Random Forest as features for XG-Boost (boosting).
- Alternatively, you can first run XG-Boost and then apply RandomForest for further refinement and correction of the errors.

ii). Advantages

- Bagging (Random Forest) helps reduce variance, preventing overfitting.
- Boosting (XG-Boost) refines the model by correcting the residuals of the previous iteration.
- This combination can handle both high variance and high bias in a more balanced manner.

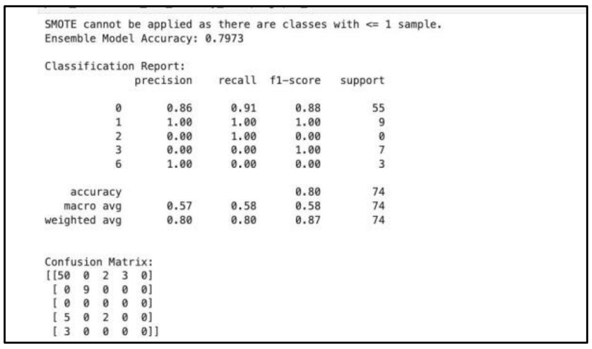
C. Benefits of Combining XG-Boost and Random Forest

- **Better Generalization**: By combining the power of bagging (Random Forest) and boosting (XG-Boost), you improve the generalization capability of the model, reducing both variance and bias.
- **Increased Accuracy**: The hybrid approach leverages the strengths of each model to increase overall prediction accuracy.
- **Reduced Overfitting**: Random Forest helps to reduce overfitting by averaging out multiple trees, while XG-Boost improves the model by focusing on difficult cases.
- **Improved Handling of Imbalanced Data**: Both models can be tuned to handle imbalanced datasets, but combining them can improve performance in classification tasks with class imbalance.
- **Increased Robustness**: The ensemble of these two models can be more robust to noise and outliers in the data.

D. Datasets

We Use CIC –IDS2017 dataset for detecting unbalanced network . We Train and Test Datasets into 80 -20 .We Clean The data So That Removes the blank and invalid data. That Improves the accuracy.

Output



```
SMOTE cannot be applied as there are classes with <= 1 sample.
Ensemble Model Accuracy: 0.7973

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.91      0.88        55
           1       1.00      1.00      1.00         9
           2       0.00      1.00      0.00         0
           3       0.00      0.00      1.00         7
           6       1.00      0.00      0.00         3

    accuracy                           0.80        74
   macro avg       0.57      0.58      0.58        74
weighted avg       0.80      0.80      0.87        74


Confusion Matrix:
[[50  0  2  3  0]
 [ 0  9  0  0  0]
 [ 0  0  0  0  0]
 [ 5  0  2  0  0]
 [ 3  0  0  0  0]]
```

Figure 6. Accuracy of Ensemble Model And Classification

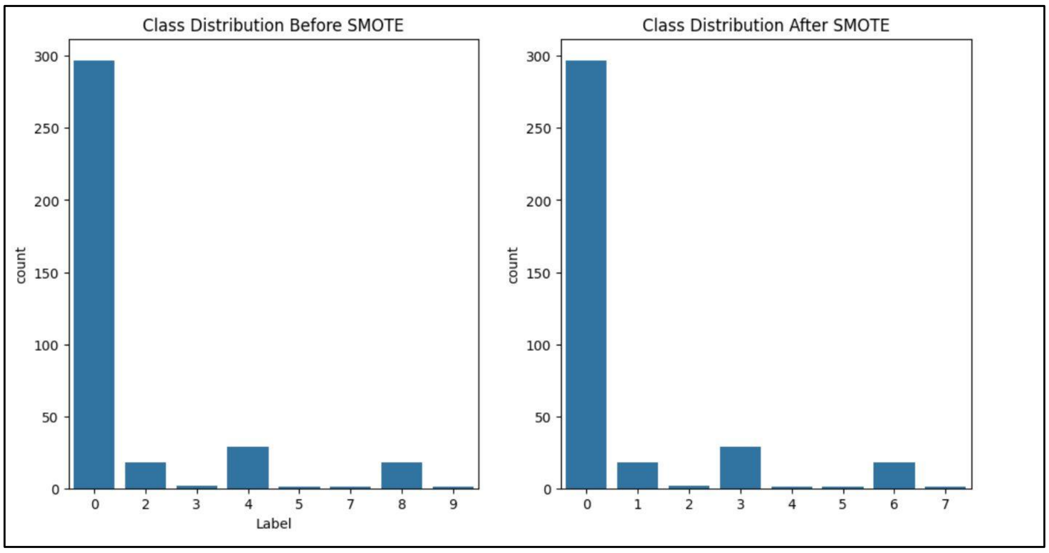
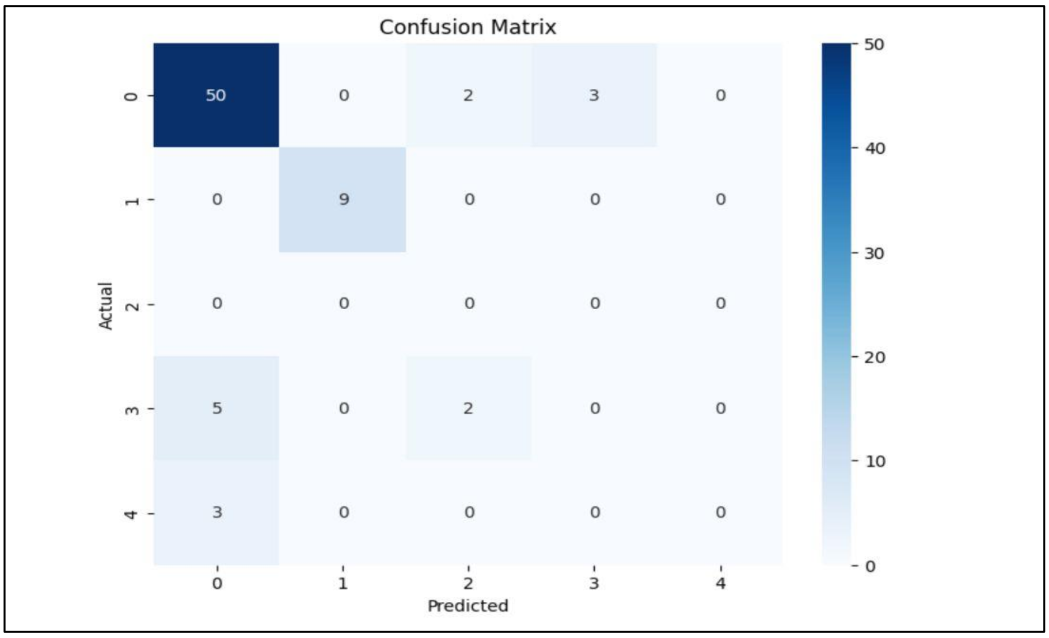


Figure 7. Class Distribution Before And After Smote



Actual \ Predicted	0	1	2	3	4
0	50	0	2	3	0
1	0	9	0	0	0
2	0	0	0	0	0
3	5	0	2	0	0
4	3	0	0	0	0

Figure 8. Confusion Matrix Of Ensemble Model

Table 1. Comparison of Individual Methods and Hybrid Method(XG-Boost + Random Forest)

Parameter	XG-Boost	Random Forest	Enhancement of Both (Hybrid)
Model Complexity	**High**: XG-Boost builds sequential trees, focusing on errors and fine-tuning them, leading to higher model complexity	**Moderate**: Random Forest builds multiple independent trees, which are simple but can grow complex with many trees	**Very High**: Combines the complexity of both models (Random Forest's bagging and XG-Boost's boosting), making it complex and harder to interpret
Interpretability	**Low**: Difficult to interpret due to the sequential nature of boosting, and interactions between	**High**: Each tree in the forest is easy to interpret, and overall predictions can be understood by	**Very Low**: Combining both models results in a highly complex structure, making it very hard to

	trees are harder to trace	looking at individual trees	interpret
Handling Nonlinear Data	**Excellent**: XG-Boost excels at capturing complex nonlinear relationships due to its boosting process and ability to model interactions between features	**Moderate**: Random Forest is better at capturing nonlinear relationships compared to a single decision tree but is less effective than XG-Boost	**Excellent**: The combination of bagging (Random Forest) and boosting (XG-Boost) provides very strong performance in handling nonlinear relationships and interactions
Scalability	**Excellent**: Highly optimized for scalability, particularly with large datasets, due to its parallel processing capabilities and efficient handling of large amounts of data	**Good**: Scales well for moderately large datasets, but can become slower with too many trees	**Excellent**: Highly optimized for scalability, particularly with large datasets, due to its parallel processing capabilities and efficient handling of large amounts of data
Overfitting Resistance	**Moderate**: XG-Boost is more prone to overfitting, especially without proper regularization, but it has built-in regularization (L1/L2) to help	**Good**: Random Forest is less prone to overfitting than a single decision tree because it averages predictions across many trees, reducing variance	**Very Good**: Combines the low-variance (Random Forest) and high-bias (XG-Boost) aspects, reducing the likelihood of overfitting and improving generalization
Training Speed	**Moderate to Slow**: Training speed can be slower due to the sequential nature of boosting, although optimizations like early stopping and parallelization can speed up training	**Moderate**: Training speed is generally slower as the number of trees increases, but parallelization helps	**Slow**: Training speed is generally slower due to the combination of both models (one for generalization and the other for error correction)
Performance on Imbalanced Data	**Excellent**: XG-Boost performs well with imbalanced datasets as it focuses on harder-to-classify examples, and class weights can be adjusted easily	**Good**: Random Forest can handle imbalanced data reasonably well by averaging predictions, but may require class weight adjustment or resampling techniques	**Excellent**: The hybrid approach handles imbalanced data well by leveraging both the robustness of Random Forest and the error-correction power of XG-Boost
Feature Interaction	**Excellent**: XG-Boost is specifically designed to capture complex feature interactions and nonlinear relationships	**Moderate**: Random Forest captures feature interactions through decision splits but is less explicit than XG-Boost	**Excellent**: The hybrid approach benefits from both models' ability to capture feature interactions-Random Forest handles the general trends, while XG-Boost refines the model through boosting
Dimensionality	**Excellent**: XG-Boost	**Good**: Can handle high-	**Excellent**: The hybrid

Handling	handles high-dimensional data well by regularizing the model and selecting the most relevant features through boosting	dimensional data, but may require feature selection techniques to avoid overfitting or inefficiency	model benefits from both approaches: Random Forest helps manage feature selection, while XGBoost focuses on the most relevant features
Robustness to Noise	**Moderate**: XGBoost can be sensitive to noise unless properly regularized, as it may overfit to noisy instances	**Good**: Random Forest is relatively robust to noisy data, as averaging predictions reduces the effect of noise	**Excellent**: The combination of both models provides a strong defense against noise, as Random Forest generalizes well and XGBoost refines the model
Practical Use	**Excellent**: Often the go-to choice for Kaggle competitions and more complex, high-accuracy tasks due to its flexibility and power.	**Good**: Works well for a variety of general-purpose tasks, particularly with structured/tabular data.	**Excellent**: Best used in high-stakes applications where both the generalization and fine-tuning abilities of the models are needed, especially in complex real-world problems.

VI. CONCLUSION

In this research, we proposed a robust ensemble model combining XG-Boost and Random Forest classifiers to detect unbalanced network traffic. By addressing the challenges of class imbalance using SMOTE, we generated synthetic samples to balance the dataset, resulting in more effective classification. The use of feature selection through Variance Thresholding and data normalization via Standard Scaling further enhanced model performance. The ensemble model demonstrated its strength by leveraging the advantages of both XG-Boost's gradient boosting approach and Random Forest's robustness to noise. Through stacked generalization, we achieved improved predictive accuracy compared to using individual classifiers. The visual analysis of the ROC curve and confusion matrix provided further insights into the model's performance, particularly in detecting minority classes. While the model showed significant improvements, minor challenges such as false positives in certain categories suggest areas for further enhancement. Future work may explore the inclusion of additional classifiers, Cine-tuning hyper parameters, or integrating real-time anomaly detection mechanisms. Overall, the proposed ensemble learning framework proves to be an effective and scalable solution for unbalanced network traffic detection, contributing to the ongoing efforts in enhancing cyber-security and mitigating network threats.

VII. REFERENCES

[1] R.R. Kumar et al., "OPTCLOUD: An Optimal Cloud Service Selection Framework Using QoS Correlation Lens," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–16, 2022. Google Scholar | Publisher Link

[2] R.R. Kumar, M. Shameem, and C. Kumar, "A Computational Framework for Ranking Prediction of Cloud Services Under Fuzzy Environment," *Enterprise Information Systems*, vol. 16, no. 1, pp. 167–187, 2022. Google Scholar | Publisher Link

[3] M. Bakro et al., "Performance Analysis Of Cloud Computing Encryption Algorithms," *Advances In Intelligent Computing And Communication*, vol. 202, pp. 357–367, 2021. Google Scholar | Publisher Link

[4] Cyber Security Market Share, Forecast, Growth Analysis, 2023.

[5] M. A. Akbar et al., "Prioritization-Based Taxonomy of Cloud-Based Outsource Software Development Challenges: Fuzzy AHP Analysis," *Applied Soft Computing*, vol. 95, 2020. Google Scholar | Publisher Link

[6] S. Lipsa, and R. K. Dash, "A Novel Dimensionality Reduction Strategy Based on Linear Regression with a Fine-Pruned Decision Tree Classifier For Detecting DDoS Attacks in Cloud Computing Environments," *Proceedings of the 1st International Symposium on Artificial Intelligence*, pp. 15–25, 2022. Google Scholar | Publisher Link

[7] N. Moustafa, and A. Jolfaei, "Autonomous Detection Of Malicious Events Using Machine Learning Models in Drone Networks," in *Proceedings of the 2nd ACM MobiCom Workshop on Drone-Assisted Wireless Communications for 5G and Beyond*, pp. 61–66, 2020. Google Scholar | Publisher Link
[8] R. Shrestha et al., "Machine Learning-Enabled Intrusion Detection System for Cellular-Connected UAV Networks," *Electronics*, vol. 10, no. 13, p. 1549, Jun. 2021. Google Scholar | Publisher Link
[9] O. Bouhamed et al., "Lightweight IDS for UAV Networks: A Periodic Deep Reinforcement Learning-Based Approach," *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 1032–1037, 2021. Google Scholar | Publisher Link
[10] J. Whelan, A. Almehmadi, and K. El-Khatib, "Artificial Intelligence for Intrusion Detection Systems in Unmanned Aerial Vehicles," *Computers and Electrical Engineering*, vol. 99, 2022. Google Scholar | Publisher Link
[11] Q. Abu Al-Haija, and A. Al Badawi, "High-Performance Intrusion Detection System for Networked UAVs Via Deep Learning," *Neural Computing and Applications*, vol. 34, no. 13, pp. 10885–10900, 2022. Google Scholar | Publisher Link
[12] E. Basan, A. Basan, A. Nekrasov, C. Fidge, E. Abramov, and A. Basyuk, "A Data Normalization Technique For Detecting Cyber Attacks On UAVs," *Drones*, vol. 6, no. 9, p. 245, Sep. 2022. Google Scholar | Publisher Link
[13] P. J. Sajith and G. Nagarajan, "Intrusion Detection System Using Deep Belief Network & Particle Swarm Optimization," *Wireless Personal Communications*, vol. 125, no. 2, pp. 1385–1403, Jul. 2022. Google Scholar | Publisher Link
[14] G. Sreelatha, A.V. Babu, and D. Midhunchakkaravarthy, "Improved Security In Cloud Using Sandpiper and Extended Equilibrium Deep Transfer Learning-Based Intrusion Detection," *Cluster Computing*, vol. 25, no. 5, pp. 3129–3144, Oct. 2022. Google Scholar | Publisher Link
[15] Q. Liu et al., "A Multi-Task Based Deep Learning Approach for Intrusion Detection," *Knowledge-Based Systems*, vol. 238, 2022. Google Scholar | Publisher Link
[16] M.A. Talukder et al., "A Dependable Hybrid Machine Learning Model for Network Intrusion Detection," *Journal of Information Security and Applications*, vol. 72, 2023. Google Scholar | Publisher Link
[17] V.B. Kamble, and N. J. Uke, *Ethical Hacking*, San International, 2024. Publisher Link
[18] V.B. Kamble et al., "Machine Learning In Fake News Detection And Social Innovation: Navigating Truth in the Digital Age," *Exploring Psychology, Social Innovation and Advanced Applications of Machine Learning*, pp. 87–108, 2025. Google Scholar | Publisher Link
[19] O. Dabade et al., "Developing An Intelligent Credit Card Fraud Detection System With Machine Learning," *Journal of Artificial Intelligence, Machine Learning and Neural Network (JAIMLNN)*, vol. 2022, ISSN 2799-1172. Google Scholar | Publisher Link
[20] V. B. Kamble et al., "Wireless Networks And Cross-Layer Design: An Implementation Approach," *International Journal of Computer Science and Information Technology (IJCSIT)*, vol. 5, no. 4, pp. 5435–5440, 2014. Google Scholar | Publisher Link
[21] V. B. Kamble, and N. J. Uke, "Image Tampering Detection: A Review of Multi-Technique Approach from Traditional to Deep Learning." *Journal of Dynamics and Control*, vol. 8, no. 11, pp. 252-283, 2024. Publisher Link
[22] A. Alshammari et al., "Classification Approach For Intrusion Detection In Vehicle Systems," *Wireless Engineering and Technology*, vol. 9, no. 4, pp. 79–94, 2018. Google Scholar | Publisher Link
[23] J. Li et al., "Machine Learning Algorithms for Network Intrusion Detection," *AI in Cybersecurity*, pp. 151–179, 2019. Google Scholar | Publisher Link
[24] K. Park, Y. Song, and Y. G. Cheong, "Classification of Attack Types for Intrusion Detection Systems Using a Machine Learning Algorithm," *Proceedings of the 2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 1-3, 2018. Google Scholar | Publisher Link
[25] S. Bernard, L. Heutte, and S. Adam, "On the Selection of Decision Trees in Random Forests," *Proceedings of the International Joint Conference on Neural Networks*, Atlanta, Georgia, USA, pp. 14–19, 2009. Google Scholar | Publisher Link
[26] A. Tesfahun, and D. L. Bhaskari, "Intrusion Detection Using Random Forests Classifier with SMOTE and Feature Reduction," *Proceedings of the 2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*. Google Scholar | Publisher Link
[27] R. Patgiri et al., "An Investigation on Intrusion Detection System Using Machine Learning." Google Scholar | Publisher Link
[28] T. S. Yange, O. Onyekware, and Y. M. Abdulmuminu, "A Data Analytics System for Network Intrusion Detection Using Decision Tree," *Journal of Computer Science and Applications*, vol. 8, pp. 21–29, 2020. Google Scholar | Publisher Link
[29] E. Hassan, M. Saleh, and A. Ahmed, "Network Intrusion Detection Approach Using Machine Learning Based on Decision Tree Algorithm," *Journal of Engineering and Applied Science*, vol. 7, pp. 1-10, 2020. Google Scholar | Publisher Link
[30] B. S. Bhati, and C. S. Rai, "Analysis of Support Vector Machine-Based Intrusion Detection Techniques," *Arabian Journal for Science and Engineering*, vol. 45, pp. 2371–2383, 2020. Google Scholar | Publisher Link

[31] Q. Shi et al., “A Framework of Intrusion Detection System Based On Bayesian Network in IoT,” *International Journal of Performability Engineering*, vol. 14, pp. 2280–2288, 2018. Google Scholar | Publisher Link

[32] M.K. Prasath and B. Perumal, “A Meta-Heuristic Bayesian Network Classification for Intrusion Detection,” *International Journal of Network Management*, vol. 29, 2019. Google Scholar | Publisher Link

[33] G. Xu, “Research on K-Nearest Neighbor High-Speed Matching Algorithm in Network Intrusion Detection,” *Netinfo Security*, vol. 20, pp. 71–80, 2020.

[34] D. Chao, Z. Gang, Y. Liu, and D. L. Zhang, “The Detection of Network Intrusion Based on Improved AdaBoost Algorithm,” *Journal of Sichuan University (Natural Science Edition)*, vol. 52, pp. 1225–1229, 2015. Google Scholar | Publisher Link

[35] K. Zhang and G. Liao, “Network Intrusion Detection Method Based on Improving Bagging-SVM Integration Diversity,” *Journal of Northeast Normal University (Natural Science Edition)*, vol. 52, pp. 53–59, 2020. Google Scholar | Publisher Link

[36] B. Li and Y. Zhang, “Research on Self-Adaptive Intrusion Detection Based on Semi-Supervised Ensemble Learning,” *Electrical Automation*, vol. 43, pp. 101–104, 2021.

[37] F. Jiang et al., “Approximate Reducts-Based Ensemble Learning Algorithm and its Application in Intrusion Detection,” *Journal of Beijing University of Technology*, vol. 42, pp. 877–885, 2016. Google Scholar | Publisher Link

[38] J. M. Xia et al., “Improved Random Forest Classifier Network Intrusion Detection Method,” *Computer Engineering and Design*, vol. 40, pp. 2146–2150, 2019. Google Scholar

[39] L. Zhang, J. Zhang, and Y. Sang, “Intrusion Detection Algorithm Based on Random Forest and Artificial Immunity,” *Computer Engineering*, vol. 46, pp. 146–152, 2020. Google Scholar

[40] J. Qiao et al., “Network Intrusion Detection Method Based on Random Forest,” *Computer Engineering and Applications*, vol. 56, pp. 82–88, 2020. Google Scholar | Publisher Link

[41] V. B. Kamble et al., “Enhancing UPI Fraud Detection: A Machine Learning Approach Using Stacked Generalization,” *International Journal of Management Science and Machine Learning (IJMSM)*, vol. 2, no. 1, pp. 69–83, 2025. Google Scholar | Publisher Link