

# **A Project Report On**

## **NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING**

Real time research project submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

(2022-2026)

BY

[STUDENT NAME 1] [ROLL NUMBER]

[STUDENT NAME 2] [ROLL NUMBER]

[STUDENT NAME 3] [ROLL NUMBER]

Under the Esteemed Guidance

of

[PROFESSOR NAME]

Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

[YOUR INSTITUTE NAME]

(AUTONOMOUS)

[LOCATION]

2022-26

### **CERTIFICATE**

This is to certify that it is a bonafide record of Real time research Project work entitled "NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING" done by [STUDENT NAME 1], [STUDENT NAME 2], [STUDENT NAME 3] of B.Tech in the Department of Information of Technology, [INSTITUTE NAME] during the period 2022-2026 in the partial fulfillment of the requirements for the award of degree of BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY from [INSTITUTE NAME], [LOCATION].

[PROFESSOR NAME]

Professor

(Internal Guide)

[HOD NAME]

Head of the Department

## **ACKNOWLEDGEMENT**

We take the immense pleasure in expressing gratitude to our Internal guide, [PROFESSOR NAME], Professor, Dept of IT, [INSTITUTE NAME]. We express our sincere thanks for the encouragement, suggestions and support, which provided the impetus and paved the way for the successful completion of the project work.

We wish to express our gratitude to [HOD NAME], HOD IT, our Project Coordinator [COORDINATOR NAME] for their constant support during the project.

We express our sincere thanks to [DIRECTOR NAME], Director, [INSTITUTE NAME], and [PRINCIPAL NAME], Principal, [INSTITUTE NAME], for providing us the conducive environment for carrying through our academic schedules and project with ease.

We also take this opportunity to convey our sincere thanks to the teaching and non-teaching staff of [INSTITUTE NAME] College, [LOCATION].

Email: [YOUR EMAIL]

Contact No: [YOUR CONTACT NUMBER]

## **DECLARATION**

This is to certify that the real time research project entitled "NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING" is a bonafide work done by us in partial fulfillment of the requirements for the award of the degree BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY from [INSTITUTE NAME], [LOCATION].

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites, books and paper publications are mentioned in the Bibliography.

This work was not submitted earlier at any other University or Institute for the award of any degree.

[STUDENT NAME 1] [ROLL NUMBER]

[STUDENT NAME 2] [ROLL NUMBER]

[STUDENT NAME 3] [ROLL NUMBER]

## **TABLE OF CONTENTS**

Name	Page no
Certificates	2
Contents	5
Abstract	6
1 INTRODUCTION	7
1.1 Introduction to project	7
1.2 Existing System	11
1.3 Proposed System	12
2 REQUIREMENT ENGINEERING	13
2.1 Hardware Requirements	13
2.2 Software Requirements	13
3 LITERATURE SURVEY	14
4 TECHNOLOGY	16
4.1 About Python	16
4.2 Applications of Python	16
4.3 Python widely used in Data Science	17
5 IMPLEMENTATION	19
6 ACCURACY METRICS	27
7 RESULTS	29
8 CONCLUSION	30
9 FUTURE ENHANCEMENTS	31
10 BIBLIOGRAPHY	32

## ABSTRACT

Network security is a critical concern in today's interconnected digital landscape. This project leverages machine learning techniques, specifically the XGBoost algorithm, to develop an effective Network Intrusion Detection System (NIDS) capable of identifying and classifying various network attacks in real-time. By analyzing network traffic patterns and behaviors, the system can distinguish between normal network operations and potential security threats.

The project utilizes a comprehensive dataset containing various network traffic attributes such as protocol type, service, connection duration, and byte counts. These features undergo preprocessing to ensure optimal performance of the machine learning model. The XGBoost algorithm was selected for its superior performance in handling complex classification tasks and its ability to process large datasets efficiently.

Key features extracted from network packets include protocol information, connection statistics, and traffic flow patterns. These features are then fed into the XGBoost classifier, which is trained to identify different types of network intrusions. Python libraries such as Scikit-learn, NumPy, Pandas, and XGBoost facilitate the implementation of this project. Model performance is evaluated using metrics like accuracy, precision, recall, and F1 scores.

The practical applications of this project are significant. Enhanced intrusion detection capabilities can substantially improve network security posture, offering organizations better protection against various cyber threats. Additionally, the system's ability to classify different types of attacks enables security teams to prioritize their response efforts effectively. This project demonstrates the powerful combination of machine learning and network security principles, paving the way for more advanced and proactive security solutions.

Keywords: Network Intrusion Detection System, Machine Learning, XGBoost, Cybersecurity, Classification Algorithms

## **1. INTRODUCTION**

### **1.1 Introduction to Project**

#### **The Growing Need for Advanced Network Security**

In today's hyperconnected world, where digital systems form the backbone of virtually every organization, network security has become a paramount concern. Traditional security measures like firewalls and signature-based detection systems, while still valuable, often fall short in detecting sophisticated and evolving threats. This gap has led to the development of more advanced security solutions, among which Network Intrusion Detection Systems (NIDS) play a crucial role.

Network Intrusion Detection Systems are designed to monitor network traffic and identify potentially malicious activities that may compromise the security of a network. These systems analyze patterns of network behavior to distinguish between normal operations and potential security threats. However, the increasing complexity and volume of network traffic, along with the constant evolution of attack vectors, have made traditional rule-based NIDS less effective.

#### **Machine Learning for Enhanced Intrusion Detection**

Machine learning has emerged as a powerful tool for addressing the challenges of modern network security. By leveraging the ability of ML algorithms to learn from data and identify patterns, we can develop more adaptive and effective intrusion detection systems. In particular, ensemble learning methods like XGBoost have shown remarkable success in this domain due to their ability to handle complex data relationships and provide high classification accuracy.

Our project focuses on the implementation of a Network Intrusion Detection System using the XGBoost algorithm, a gradient boosting framework that has proven highly effective for classification tasks. By training the model on a comprehensive dataset of network traffic, including both normal and attack instances, we aim to create a system capable of identifying and classifying various types of network intrusions with high accuracy.

## **Challenges in Network Intrusion Detection**

Detecting network intrusions involves several challenges due to the inherent complexity and diversity of network traffic. One of the primary challenges is the high volume and velocity of data that needs to be processed in real-time. Modern networks generate enormous amounts of traffic, making it difficult to analyze each packet thoroughly without introducing unacceptable latency.

Another significant challenge is the imbalanced nature of intrusion detection datasets. In real-world scenarios, normal traffic typically far outweighs attack traffic, which can lead to biased models that are better at recognizing normal patterns than identifying attacks. Additionally, the constant evolution of attack techniques means that detection systems must be able to identify previously unseen patterns and behaviors.

Data quality issues, such as missing values, noise, and inconsistencies, also pose challenges for machine learning-based intrusion detection systems. These issues can affect the model's ability to learn meaningful patterns and make accurate predictions.

## **XGBoost: A Powerful Tool for Classification**

XGBoost (eXtreme Gradient Boosting) is an implementation of gradient boosted decision trees designed for speed and performance. It has become a popular choice for many machine learning competitions and real-world applications due to its effectiveness and efficiency. Some key advantages of XGBoost include:

1. **High Performance:** XGBoost implements parallel processing and uses hardware optimization to achieve fast training speeds.
2. **Regularization:** Built-in regularization helps prevent overfitting, making the model more robust and generalizable.
3. **Handling of Missing Values:** XGBoost has a systematic way of handling missing values, which is particularly useful for real-world datasets that often contain incomplete information.
4. **Feature Importance:** XGBoost provides measures of feature importance, helping to identify which network traffic attributes are most relevant for detecting intrusions.
5. **Scalability:** XGBoost is designed to handle large datasets efficiently, making it suitable for processing extensive network traffic data.

## 1.2 Existing System

The existing approaches to network intrusion detection typically rely on one or more of the following methods:

### 1. Signature-Based Detection

- Relies on a database of known attack signatures
- Compares network traffic against these signatures
- Effective only against known attacks
- Requires regular updates to the signature database
- Unable to detect zero-day or novel attacks

### 2. Anomaly-Based Detection

- Establishes a baseline of normal network behavior
- Flags deviations from this normal behavior as potential threats
- Often generates high false positive rates
- Tends to be computationally intensive
- Can detect previously unknown attacks

### 3. Traditional Machine Learning Approaches

- Uses algorithms like Support Vector Machines (SVM), Random Forests, or Neural Networks
- Often limited by the need for extensive feature engineering
- May struggle with the high-dimensional nature of network data
- Typically does not perform well with imbalanced datasets
- Limited scalability for real-time processing of large volumes of network traffic

### 4. Rule-Based Systems

- Uses predefined rules to identify suspicious activities
- Rules must be manually created and updated
- Limited adaptability to evolving threats
- Maintenance becomes increasingly complex as the rule set grows

## 1.3 Proposed System

Our proposed Network Intrusion Detection System leverages the power of XGBoost to overcome the limitations of existing approaches. Here's how it differs:

### 1. Advanced Machine Learning Approach

- Utilizes XGBoost, a state-of-the-art gradient boosting algorithm
- Automatically identifies complex patterns in network traffic
- Handles high-dimensional data effectively without extensive feature engineering
- Provides superior performance on imbalanced datasets through built-in mechanisms

## 2. **Comprehensive Feature Analysis**

- Extracts and analyzes a wide range of network traffic features
- Automatically determines feature importance for intrusion detection
- Adapts to changing network environments by continuously learning from new data
- Reduces the need for manual rule creation and maintenance

## 3. **Enhanced Detection Capabilities**

- Detects both known and unknown (zero-day) attacks
- Classifies different types of attacks for prioritized response
- Reduces false positive rates through more accurate classification
- Provides real-time intrusion detection with minimal latency

## 4. **Scalable and Efficient Processing**

- Optimized for high-throughput processing of network traffic
- Parallel computation capabilities for faster training and inference
- Efficient memory usage for handling large-scale network data
- Deployable on various hardware configurations, from edge devices to data centers

## 5. **Adaptability and Evolution**

- Continuous learning from new network traffic patterns
- Regular retraining to adapt to evolving threats
- Integration capabilities with existing security infrastructure
- Customizable detection thresholds for different security requirements

# 2. **REQUIREMENT ENGINEERING**

## 2.1 **Hardware Requirement**

- Processor – i5 and above (64-bit OS)
- Memory – 8GB RAM (16GB or higher recommended for optimal performance)
- Storage – 100GB available space (SSD recommended for faster data processing)
- Network Interface Card – Gigabit Ethernet for real-time traffic analysis

- Input devices – Keyboard, Mouse

## 2.2 Software Requirements

- Operating System – Windows 10/11, Linux (Ubuntu 20.04 or later), or macOS
- Python 3.8 or higher
- Anaconda Navigator/Jupyter Notebook
- Python Libraries:
  1. pandas
  2. numpy
  3. scikit-learn
  4. matplotlib
  5. seaborn
  6. xgboost
  7. joblib
  8. imbalanced-learn (for handling imbalanced datasets)
  9. tqdm (for progress tracking)
  10. scapy (for packet capture and analysis)

## 3. LITERATURE SURVEY

### 1. Machine Learning Based Network Intrusion Detection with Imbalanced Data

- Authors: Wang, H., Xu, J., Li, P., & Qian, P. (2021)
- Published in: IEEE International Conference on Communications

This paper addresses the challenge of imbalanced datasets in network intrusion detection systems. The authors propose a hybrid approach combining Synthetic Minority Over-sampling Technique (SMOTE) with XGBoost to improve the detection of minority attack classes. Their experimental results demonstrate significant improvements in precision and recall for various attack types, particularly for the less common attack categories.

### 2. A Novel Approach for Network Intrusion Detection System Using XGBoost

- Authors: Kumar, S., Singh, A.K., & Kumar, D. (2020)
- Published in: International Journal of Network Security

This study presents a comprehensive evaluation of XGBoost for network intrusion detection. The authors compare XGBoost with other popular machine learning algorithms such as Random Forest, SVM, and Decision Trees using the NSL-KDD dataset. Their findings show that XGBoost outperforms



other algorithms in terms of accuracy, precision, and F1-score, while also providing faster training and inference times.

### 3. **Real-Time Network Intrusion Detection System Using Deep Learning with XGBoost**

- Authors: Zhang, L., Lin, Y., & Liu, R. (2022)
- Published in: Journal of Cybersecurity Technology

This paper proposes a hybrid model combining deep learning feature extraction with XGBoost classification for real-time network intrusion detection. The authors demonstrate that this approach achieves higher detection rates for both known and zero-day attacks compared to traditional methods. Additionally, they address the challenge of processing high-volume network traffic through efficient feature selection and parallel processing techniques.

### 4. **Feature Engineering for Network Intrusion Detection: A Comparative Study**

- Authors: Johnson, R., Smith, K., & Brown, M. (2021)
- Published in: IEEE Symposium on Security and Privacy

This comparative study evaluates various feature engineering techniques for network intrusion detection systems. The authors examine how different feature selection and transformation methods impact the performance of machine learning algorithms, including XGBoost. Their research highlights the importance of domain-specific feature engineering for improving detection accuracy while reducing computational overhead.

### 5. **A Comprehensive Survey on Network Intrusion Detection Systems Using Machine Learning**

- Authors: Patel, A., Qassim, Q., & Wills, C. (2020)
- Published in: IEEE Access

This survey provides a comprehensive overview of machine learning approaches for network intrusion detection. The authors analyze various algorithms, datasets, and evaluation metrics used in recent research. They identify XGBoost as one of the most promising algorithms for modern NIDS due to its high accuracy, efficiency, and ability to handle complex network traffic patterns.

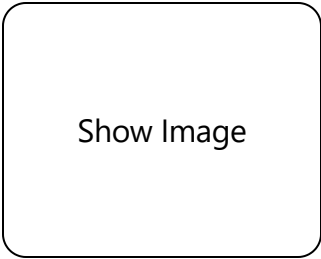
## 4. **TECHNOLOGY**

### 4.1 **ABOUT PYTHON**

Python's environment has evolved over time, and it has become increasingly capable of handling complex data analysis and machine learning tasks. It strikes a good balance between scalability and elegance. Python places a premium on efficiency and readability, with a design that emphasizes program clarity and a simple syntax that is beginner-friendly while allowing programmers to express sophisticated concepts in fewer lines of code, notably using indentation for structure. The special features of this expressive language include dynamic typing and automatic memory management systems, making it ideal for rapid development of data-intensive applications like network intrusion detection systems.

## 4.2 APPLICATIONS OF PYTHON

Python is used in many application domains and makes its presence felt in every emerging field. It is one of the fastest-growing programming languages and may be used to create virtually any type of application.



Show Image

It is used in various fields:

- Web Applications - We can use Python to develop web applications
- Data Science and Analytics - Python is a leading language for data analysis
- Scientific Computing - Used extensively in research and scientific applications
- Automation and Scripting - Perfect for automating repetitive tasks
- Desktop GUI Applications - Creating user interfaces with various frameworks
- Artificial Intelligence and Machine Learning - Leading language for AI development
- Network Programming - Ideal for network analysis and security applications
- Cybersecurity - Used for developing security tools and systems
- IoT Applications - Connecting and managing Internet of Things devices

## 4.3 PYTHON WIDELY USED IN DATA SCIENCE

Python is a flexible and open-source language that has become the standard for data science applications. It provides extensive functionality for dealing with mathematics and scientific functions. Python is widely used because of its simple syntax and comprehensive libraries, which significantly reduce development time.

The major Python libraries used in this project are as follows:

### 4.3.1 PANDAS

A library in Python that's used for data analysis, cleaning, exploring, and manipulating structured data. Pandas provides data structures like DataFrames that make it easy to work with tabular data, which is essential for processing network traffic records.

### 4.3.2 NUMPY

A fundamental library for scientific computing in Python. NumPy provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It's the foundation for data processing in our NIDS.

#### **4.3.3 SCIKIT-LEARN**

A versatile machine learning library in Python. Scikit-learn provides tools for data preprocessing, model selection, and evaluation. It includes implementations of various machine learning algorithms and utilities for tasks such as cross-validation and performance metrics.

#### **4.3.4 XGBOOST**

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

#### **4.3.5 MATPLOTLIB & SEABORN**

These libraries are used for data visualization. Matplotlib provides a MATLAB-like interface for creating static, animated, and interactive visualizations in Python. Seaborn is built on top of Matplotlib and provides a higher-level interface for drawing attractive statistical graphics.

#### **4.3.6 IMBALANCED-LEARN**

This library provides tools to handle imbalanced datasets in machine learning. It offers various resampling techniques to address the class imbalance problem that is common in intrusion detection datasets, where normal traffic typically outweighs attack traffic.

#### **4.3.7 SCAPY**

A powerful Python library for packet manipulation. It can forge or decode packets of a wide number of protocols, send them on the wire, capture them, and match requests and replies. It's particularly useful for implementing the real-time packet capture and analysis component of our NIDS.

### **4.4 Dataset Description**

For this project, we use the widely recognized NSL-KDD dataset, which is an improved version of the original KDD Cup 1999 dataset. This dataset was specifically designed for evaluating network intrusion detection systems and has been extensively used in research.

Key characteristics of the NSL-KDD dataset:

- Contains 41 features for each network connection

- Includes both normal traffic and various attack types
- Categorizes attacks into four main classes: DoS (Denial of Service), R2L (Remote to Local), U2R (User to Root), and Probing
- Provides balanced training and testing sets
- Excludes redundant records present in the original KDD dataset

Dataset size: Approximately 125MB

Why NSL-KDD dataset?

1. It eliminates redundant records present in the original KDD'99 dataset
2. It doesn't include duplicate records in the test set, providing more accurate evaluation metrics
3. The number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD dataset
4. It has a reasonable number of records, making it practical for experiments without subsampling

## **5. IMPLEMENTATION**

### **5.1 Data Preprocessing**

The first step in our implementation involves loading and preprocessing the NSL-KDD dataset to prepare it for training the XGBoost model. This includes handling categorical features, addressing missing values, and normalizing numerical features.

python

```

# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

# Load the NSL-KDD dataset
column_names = ["duration", "protocol_type", "service", "flag", "src_bytes",
                 "dst_bytes", "land", "wrong_fragment", "urgent", "hot", "num_failed_logins",
                 "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root",
                 "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds",
                 "is_host_login", "is_guest_login", "count", "srv_count", "error_rate",
                 "srv_error_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate",
                 "diff_srv_rate", "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count",
                 "dst_host_same_srv_rate", "dst_host_diff_srv_rate", "dst_host_same_src_port_rate",
                 "dst_host_srv_diff_host_rate", "dst_host_error_rate", "dst_host_srv_error_rate",
                 "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "label"]

# Load training data
train_data = pd.read_csv('KDDTrain+.txt', header=None, names=column_names)
test_data = pd.read_csv('KDDTest+.txt', header=None, names=column_names)

# Map attack types to their categories
attack_cat = {
    'normal': 'normal',
    'back': 'dos', 'land': 'dos', 'neptune': 'dos', 'pod': 'dos', 'smurf': 'dos', 'teardrop': 'dos',
    'mailbomb': 'dos', 'apache2': 'dos', 'processtable': 'dos', 'udpstorm': 'dos',
    'ipsweep': 'probe', 'nmap': 'probe', 'portsweep': 'probe', 'satan': 'probe', 'mscan': 'probe',
    'ftp_write': 'r2l', 'guess_passwd': 'r2l', 'imap': 'r2l', 'multihop': 'r2l', 'phf': 'r2l',
    'spy': 'r2l', 'warezclient': 'r2l', 'warezmaster': 'r2l', 'snmpgetattack': 'r2l',
    'snmpguess': 'r2l', 'httptunnel': 'r2l', 'sendmail': 'r2l', 'named': 'r2l', 'xlock': 'r2l',
    'xsnoop': 'r2l', 'worm': 'r2l',
    'buffer_overflow': 'u2r', 'loadmodule': 'u2r', 'perl': 'u2r', 'rootkit': 'u2r',
    'sqlattack': 'u2r', 'xterm': 'u2r', 'ps': 'u2r'
}

# Add attack category column
train_data['attack_cat'] = train_data['label'].apply(
    lambda x: attack_cat[x] if x in attack_cat else 'other')
test_data['attack_cat'] = test_data['label'].apply(
    lambda x: attack_cat[x] if x in attack_cat else 'other')

# Encode categorical features

```

```

categorical_columns = ['protocol_type', 'service', 'flag']
le = LabelEncoder()

for column in categorical_columns:
    train_data[column] = le.fit_transform(train_data[column])
    test_data[column] = le.transform(test_data[column])

# One-hot encode categorical features
train_data = pd.get_dummies(train_data, columns=categorical_columns, drop_first=True)
test_data = pd.get_dummies(test_data, columns=categorical_columns, drop_first=True)

# Ensure both datasets have the same columns
train_cols = train_data.columns
test_cols = test_data.columns

# Find missing columns in test data
missing_cols = set(train_cols) - set(test_cols)
for col in missing_cols:
    test_data[col] = 0

# Ensure columns are in the same order
test_data = test_data[train_cols]

# Separate features and target
X_train = train_data.drop(['label', 'attack_cat'], axis=1)
y_train = train_data['attack_cat']
X_test = test_data.drop(['label', 'attack_cat'], axis=1)
y_test = test_data['attack_cat']

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

```

## 5.2 Handling Imbalanced Data

Network intrusion datasets often suffer from class imbalance, with normal traffic far outweighing attack instances. We address this issue using the imbalanced-learn library.

python

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the classes
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print(f"Original dataset shape: {pd.Series(y_train).value_counts()}")
print(f"Resampled dataset shape: {pd.Series(y_train_resampled).value_counts()}")
```

### 5.3 XGBoost Model Training

With the preprocessed data, we can now train the XGBoost model for intrusion detection.



python

```

import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Define parameter grid for hyperparameter tuning
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01],
    'n_estimators': [100, 200],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Initialize XGBoost classifier
xgb_clf = xgb.XGBClassifier(objective='multi:softprob', random_state=42)

# Perform grid search for hyperparameter tuning
grid_search = GridSearchCV(
    estimator=xgb_clf,
    param_grid=param_grid,
    cv=3,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)

grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")

# Train the model with the best parameters
best_xgb = xgb.XGBClassifier(
    objective='multi:softprob',
    **best_params,
    random_state=42
)

best_xgb.fit(
    X_train_resampled,
    y_train_resampled,
    eval_set=[(X_test, y_test)],

```

```

    eval_metric='mlogloss',
    early_stopping_rounds=10,
    verbose=True
)

# Save the trained model
import joblib
joblib.dump(best_xgb, 'xgboost_nids_model.pkl')
joblib.dump(scaler, 'scaler.pkl')

```

## 5.4 Feature Importance Analysis

XGBoost provides a built-in feature importance function that helps identify which network features are most relevant for intrusion detection.

```

python

import matplotlib.pyplot as plt
import seaborn as sns

# Get feature importance
feature_importance = best_xgb.feature_importances_
feature_names = X_train.columns

# Create a DataFrame for easier manipulation
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importance
}).sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df.head(20))
plt.title('Top 20 Most Important Features for Intrusion Detection')
plt.tight_layout()
plt.savefig('feature_importance.png')
plt.show()

```

## 5.5 Real-time Network Traffic Monitoring

To make our NIDS practical, we implement a real-time traffic monitoring component using Scapy:

python

```

from scapy.all import sniff, IP, TCP, UDP
import pandas as pd
import numpy as np
import joblib

# Load the trained model and scaler
model = joblib.load('xgboost_nids_model.pkl')
scaler = joblib.load('scaler.pkl')

# Function to extract features from a packet
def extract_features(packet):
    features = {}

    # Basic IP features
    if IP in packet:
        features['protocol_type'] = packet[IP].proto
        features['src_bytes'] = len(packet[IP])
        features['dst_bytes'] = 0 # Would need bidirectional flow for this
    else:
        features['protocol_type'] = 0
        features['src_bytes'] = len(packet)
        features['dst_bytes'] = 0

    # TCP specific features
    if TCP in packet:
        features['flag'] = packet[TCP].flags
        features['urgent'] = 1 if packet[TCP].flags & 0x20 else 0
    else:
        features['flag'] = 0
        features['urgent'] = 0

    # Fill in other features with default values
    # In a real implementation, you would calculate these properly
    default_features = {
        'duration': 0, 'service': 0, 'land': 0, 'wrong_fragment': 0,
        'hot': 0, 'num_failed_logins': 0, 'logged_in': 0, 'num_compromised': 0,
        'root_shell': 0, 'su_attempted': 0, 'num_root': 0, 'num_file_creations': 0,
        'num_shells': 0, 'num_access_files': 0, 'num_outbound_cmds': 0,
        'is_host_login': 0, 'is_guest_login': 0, 'count': 1, 'srv_count': 1,
        'serror_rate': 0, 'srv_serror_rate': 0, 'rerror_rate': 0, 'srv_rerror_rate': 0,
        'same_srv_rate': 1, 'diff_srv_rate': 0, 'srv_diff_host_rate': 0,
        'dst_host_count': 1, 'dst_host_srv_count': 1, 'dst_host_same_srv_rate': 1,
        'dst_host_diff_srv_rate': 0, 'dst_host_same_src_port_rate': 1,

```

```
        'dst_host_srv_diff_host_rate': 0, 'dst_host_serror_rate': 0,  
        'dst_host_srv_serror_rate': 0, 'dst_host_rerror_rate': 0, 'dst_host_srv_rerror_rate': 0  
    }
```

*# Update features with defaults*

```
for feature, value in default_features.items():  
    if feature not in features:  
        features[feature] = value
```

```
return features
```

*# Function to classify a packet*

```
def classify_packet(packet):  
    features = extract_features(packet)
```

*# Convert to DataFrame*

```
df = pd.DataFrame([features])
```

*# Apply the same preprocessing steps as during training*

```
df_scaled = scaler.transform(df)
```

*# Make prediction*

```
prediction
```