A

Project Report on

Intrusion Detection System using Machine Learning

Submitted in partial fulfilment for the award of the degree of Bachelor of Technology

in

Computer Science & Engineering

Submitted By:

Supervised by

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Mission Statement

The mission of this project is to design, develop, and evaluate an intelligent Intrusion Detection System (IDS) utilizing machine learning algorithms to enhance the security posture of computer networks against cyberattacks. By leveraging the capabilities of Random Forest and Support Vector Machine (SVM) classifiers on the NSL-KDD dataset, the system aims to accurately detect various forms of network intrusions—ranging from Denial of Service (DoS) to user privilege escalation attacks—while minimizing false alarms and optimizing computational efficiency. The broader objective is to demonstrate the practical applicability of machine learning in modern cybersecurity defense strategies and to contribute to the ongoing evolution of intelligent threat detection systems.

1.2 Objectives

The objectives of this project are multifaceted, encompassing technical implementation, evaluation, and knowledge contribution. The primary objectives include:

- To understand and analyze the limitations of traditional IDS techniques, particularly in detecting unknown and sophisticated threats.
- To explore the application of supervised machine learning algorithms—specifically Random Forest and SVM—for classifying and detecting network intrusions.
- To implement a working prototype of an IDS using the NSL-KDD dataset, including data preprocessing, model training, and evaluation.
- To compare the performance of the two machine learning models based on accuracy, precision, recall, and F1-score.
- To provide data visualizations that offer insight into feature distributions, attack patterns, and classifier performance.
- To identify and address challenges such as dataset imbalance, noise, and zero-day detection within the scope of the project.
- To propose potential enhancements and future directions for real-world deployment and scalability.

1.3 Project Goals

The project aims to meet the following specific goals:

- **Model Accuracy:** Achieve a detection accuracy of at least 90% for distinguishing between normal and malicious traffic across multiple attack types in the NSL-KDD dataset.
- **Model Comparison:** Provide a detailed comparative analysis between Random Forest and SVM models, highlighting their respective strengths and limitations in intrusion detection scenarios.
- **Feature Engineering:** Apply appropriate feature selection and transformation techniques to improve classifier performance and training efficiency.
- **Usability and Interpretability:** Ensure that the implemented IDS is not only effective but also interpretable, providing users with actionable insights on detected threats.

- **Educational Value:** Serve as a comprehensive academic project that demonstrates practical skills in cybersecurity, machine learning, and data analysis, while also aligning with current industry demands and research trends.
- **Scalability Awareness:** While the prototype is based on the NSL-KDD dataset, the design approach is intended to be scalable to larger and real-time datasets such as CICIDS2017 or ToN-IoT.

1.4 Features

This project incorporates several essential features designed to simulate a real-world Intrusion Detection System while keeping the scope focused and educational:

- **Machine Learning-Based Detection:** Utilizes two powerful supervised algorithms—Random Forest and SVM—for detecting intrusions.
- **Preprocessing Pipeline:** Implements data normalization, label encoding, and dimensionality reduction to optimize model training.
- **Model Evaluation:** Provides detailed evaluation metrics (accuracy, precision, recall, F1-score, and confusion matrix) to assess model effectiveness.
- **Visualization Dashboard:** Includes graphs and charts (e.g., bar plots, heatmaps) to visualize feature importance, class distributions, and performance metrics.
- **Support for Multi-Class Classification:** Capable of identifying multiple types of intrusions such as DoS, Probe, R2L, and U2R attacks.
- **Error Analysis:** Includes analysis of false positives and false negatives to understand model behavior under different conditions.
- **Research Integration:** References and briefly discusses additional datasets (e.g., CICIDS2017, ToN-IoT) to contextualize the applicability and relevance of the project.

1.5 Advantages

The use of machine learning in this IDS implementation offers several key advantages over traditional, signature-based detection systems:

- **Improved Accuracy:** Machine learning models learn complex patterns in network traffic, leading to better accuracy in intrusion detection.
- **Zero-Day Threat Detection:** Unlike signature-based systems, ML-based IDS can identify previously unseen attack types by recognizing anomalies and learned behavior.
- **Automation:** The detection process is automated, requiring minimal human intervention after deployment.
- **Scalability:** ML-based solutions are inherently scalable and adaptable to large datasets and real-time environments.
- **Customizability:** Models can be retrained and fine-tuned with new data to keep up with evolving cyber threats.
- **Insightful Analytics:** The use of visualizations and statistical summaries allows better understanding and decision-making for security analysts.

1.6 Project Outputs

The outputs of this project include both tangible deliverables and academic contributions:

- **Fully Functional IDS Prototype:** A working intrusion detection prototype using Python and Jupyter Notebook, capable of analyzing the NSL-KDD dataset and predicting intrusions with high accuracy.
- **Source Code and Documentation:** Clean, well-commented code with accompanying documentation covering installation, usage instructions, and model configurations.
- **Performance Reports:** Tabular and graphical reports of model performance on test data, along with detailed analysis and interpretation.
- **Visualization Modules:** Charts and graphs illustrating dataset properties, model performance, and feature importance for deeper understanding.
- **Research Report:** A detailed project report structured across chapters (Introduction, Literature Review, Methodology, Results, Conclusion, etc.), formatted for academic submission.
- **Recommendations and Future Scope:** Suggestions for future improvements, such as integrating deep learning models (e.g., LSTM, CNN), deploying in a real-time environment, or expanding to newer datasets like CICIDS2017 or UNSW-NB15.

Chapter 2: Literature Review

2.1 IDS Background

Intrusion Detection Systems (IDS) have long been a cornerstone of network and system security, serving as the first line of defense against unauthorized access and cyberattacks. Their primary function is to continuously monitor system or network activities to identify anomalies, potential breaches, or malicious behaviors. IDS solutions can be broadly classified based on their deployment location and detection strategy.

Types of IDS:

- **Network-Based IDS (NIDS):** These systems are strategically positioned within network infrastructure to analyze traffic flows in real-time. They inspect data packets moving across the network and are particularly effective for large-scale environments such as enterprise networks or data centers. NIDS can detect a wide variety of attacks, including port scans, denial-of-service (DoS) attempts, and protocol-based exploits.
- **Host-Based IDS (HIDS):** Deployed directly on individual devices, HIDS focus on internal system metrics such as log files, file integrity, system calls, and user behavior. These systems are tailored for endpoint security and offer a detailed, localized view of potential intrusions. While they provide higher accuracy in detecting attacks at the host level, they require more computational resources and may be difficult to scale across large infrastructures.

Detection Techniques in IDS:

- **Signature-Based Detection:** This technique relies on a predefined database of known attack patterns or signatures. It is highly effective in identifying known threats with high accuracy and low false positives. However, it is limited by its inability to detect novel or zero-day attacks, which do not match existing signatures.
- **Anomaly-Based Detection:** This method involves establishing a baseline of "normal" behavior for a system or network. Any significant deviation from this baseline is flagged as a potential threat. Anomaly-based detection is capable of identifying new or previously unseen attacks, making it suitable for dynamic threat environments. However, it often suffers from a higher false-positive rate due to the difficulty in accurately modeling "normal" behavior.

Modern cybersecurity strategies often combine both detection techniques in hybrid IDS solutions to leverage the strengths of each.

2.2 ML Integration in IDS

With the increasing complexity of cyber threats, traditional IDS approaches have become insufficient in isolation. Machine Learning (ML) offers a powerful alternative by introducing adaptability, automation, and intelligence into the detection process. ML-based IDS can identify intricate attack patterns, reduce human intervention, and improve the accuracy of threat classification.

Types of Machine Learning Techniques in IDS:

- **Supervised Learning:** In supervised learning, models are trained on labeled datasets where each instance is marked as either benign or malicious. Common algorithms include:
    - **Random Forest:** An ensemble-based model that builds multiple decision trees and aggregates their outputs. It is robust to overfitting and effective for multiclass classification tasks.
    - **Support Vector Machine (SVM):** A powerful algorithm that constructs optimal hyperplanes to separate different classes. It works well with highdimensional data and is particularly effective for binary classification.
    - **Decision Trees and Naive Bayes:** Frequently used for their interpretability and simplicity in smaller datasets.

  Supervised learning performs well in scenarios where labeled data is abundant and representative of real-world conditions.

- **Unsupervised Learning:** In situations where labeled data is unavailable, unsupervised learning methods are used to uncover hidden patterns and anomalies. Algorithms like **K-Means clustering**, **DBSCAN**, and **Autoencoders** help detect outliers that may represent intrusions. While less accurate than supervised learning in structured tasks, these models are valuable for discovering zero-day and insider threats.
- **Deep Learning:** A subset of ML that uses neural networks to model highly complex, non-linear relationships. Techniques such as:
    - **Convolutional Neural Networks (CNN):** Effective for detecting spatial features in packet-level data.
    - **Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM):** Useful for modeling sequential data, such as time-based patterns in network traffic.

Deep learning models require large datasets and computational resources but can outperform traditional algorithms in complex scenarios.

Benefits of ML in IDS:

- **Adaptability:** Models can be retrained on new data to stay current with emerging attack techniques.
- **Efficiency:** Capable of processing vast amounts of data at high speed, suitable for real-time monitoring.
- **Improved Detection Rates:** Reduces false negatives and improves the identification of subtle or sophisticated intrusions.
- **Automation:** Minimizes manual rule creation, allowing for dynamic response to threats.

2.3 Dataset Overview

The quality and structure of the dataset used for training ML models play a crucial role in the performance of an IDS. Several benchmark datasets are widely used in the research community:

- **NSL-KDD:** An improved version of the KDD'99 dataset, NSL-KDD addresses issues such as redundancy and class imbalance. It includes a labeled set of network traffic data categorized into normal activity and various attack types (DoS, R2L, U2R, Probe). Its reduced complexity and manageable size make it ideal for academic experimentation.
- **CICIDS2017:** This modern dataset simulates real-world traffic, including benign and malicious flows, captured from a network testbed. It covers a wide range of attack scenarios and includes over 80 features extracted using packet analyzers.
- **ToN-IoT:** A dataset designed specifically for Internet of Things (IoT) environments, incorporating telemetry and device-level attack data. It supports IDS research in emerging IoT contexts.

For this project, **NSL-KDD** was chosen due to its accessibility, structured labeling, and alignment with academic standards for supervised learning tasks.

2.4 Related Works

Numerous studies have explored the integration of machine learning in IDS. In [1], a Random Forest-based model achieved over 93% accuracy in detecting KDD dataset attacks, demonstrating ensemble learning's robustness. Similarly, SVMs have shown effectiveness in binary intrusion classification tasks, particularly when paired with kernel functions like RBF for handling non-linear relationships.

Recent research trends are shifting towards hybrid models and deep learning architectures. A 2023 study applied a hybrid CNN-LSTM model to the CICIDS2017 dataset, achieving stateof-the-art detection rates for DoS and Brute Force attacks [2]. Other works focus on improving feature selection using techniques like PCA, mutual information, and genetic algorithms to optimize model training.

While many of these approaches achieve high accuracy in controlled settings, challenges such as data imbalance, overfitting, and real-time deployment remain areas of active exploration. This project builds upon these foundations by comparing classic supervised models (Random Forest and SVM) and addressing practical issues in feature selection, model evaluation, and future scalability.

Chapter 3: System Design

3.1 Workflow & Use Case Diagram

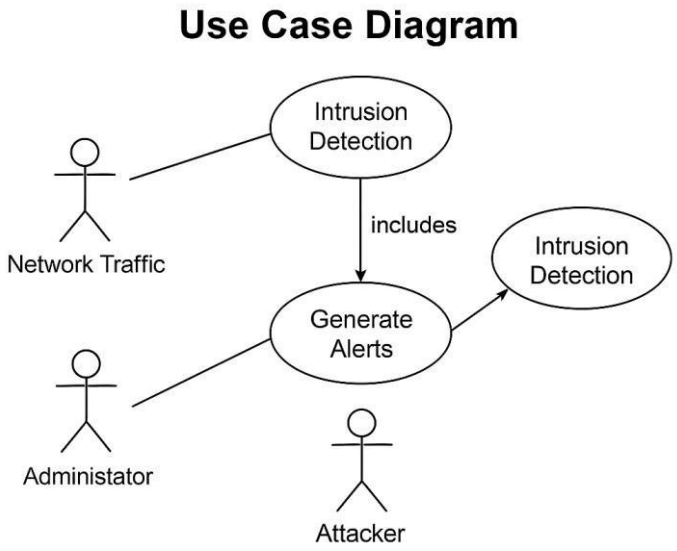
The system design of a Machine Learning-based Intrusion Detection System (IDS) involves a clear workflow that transitions from data acquisition to model evaluation and prediction. The use case primarily involves a network administrator or security analyst who uses the system to detect potential intrusions in network traffic.

Workflow Steps:

1. **Data Collection:** Ingest NSL-KDD dataset or other benchmark datasets.
2. **Data Preprocessing:** Clean and normalize data, handle missing values, encode categorical variables.
3. **Feature Selection:** Use techniques like correlation analysis or recursive feature elimination.
4. **Model Training:** Train ML models (Random Forest, SVM, etc.) using training data.

5. **Model Evaluation:** Evaluate models using test data, visualize metrics such as accuracy, precision, recall, and confusion matrix.
6. **Intrusion Detection:** Deploy trained model for real-time or batch detection.

Fig.1: Use Case Diagram



Actors:

- Admin
- IDS System **Use Cases:**

- Upload Dataset
- Train Model
- Detect Intrusion
- View Results

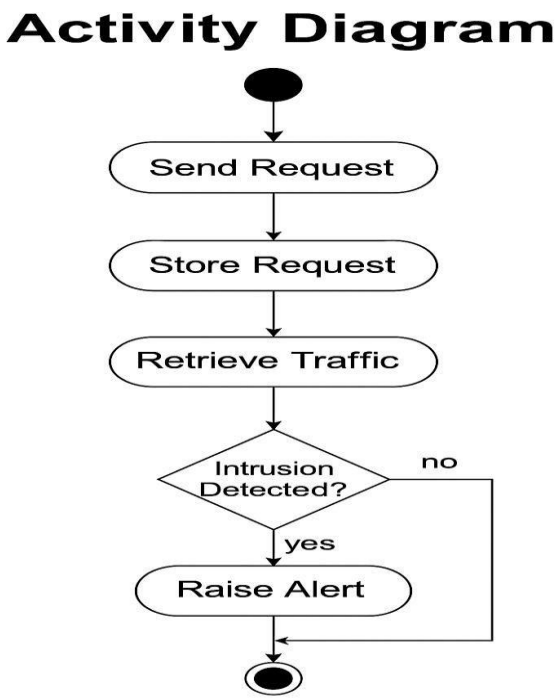
3.2 Activity Diagram

The activity diagram illustrates the sequential flow of actions taken during the IDS lifecycle:

1. Start
2. Load Dataset
3. Preprocess Data
4. Select Features
5. Choose ML Algorithm
6. Train Model
7. Evaluate Model
8. Deploy Model
9. Detect Intrusions

10. Display Alerts/Reports
11. End

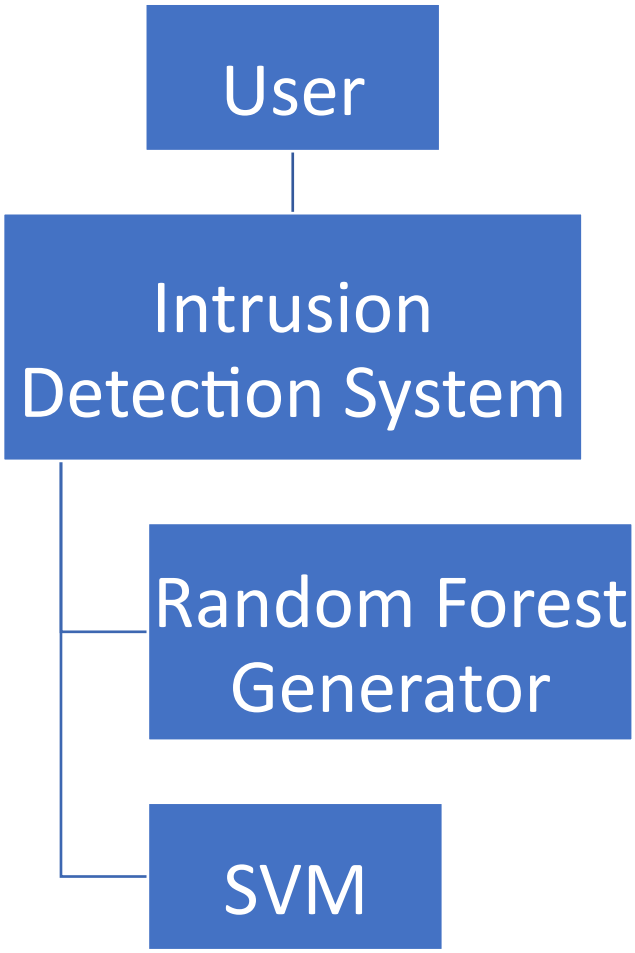
Fig.2: Activity Diagram



3.3 System Architecture

The architecture of the IDS is modular and comprises the following layers:

1. **Data Layer:** Responsible for collecting and storing raw input data (NSL-KDD dataset).
2. **Processing Layer:** Handles preprocessing tasks such as data cleaning, normalization, and feature selection.
3. **Model Layer:** Core layer where ML models (Random Forest, SVM) are trained and tested.
4. **Detection Layer:** Applies the trained models on incoming data to detect anomalies or intrusions.
5. **Visualization Layer:** Presents results via graphs and dashboards to end-users.



3.4 Data Flow Diagram (Level 1)

Entities:

- Admin/User
- IDS System

Processes:

1. Data Input
2. Preprocessing
3. Feature Selection
4. Model Training
5. Intrusion Detection

Data Stores:

- Dataset Repository
- Model Repository
- Detection Logs

3.5 Component Diagram

Main Components:

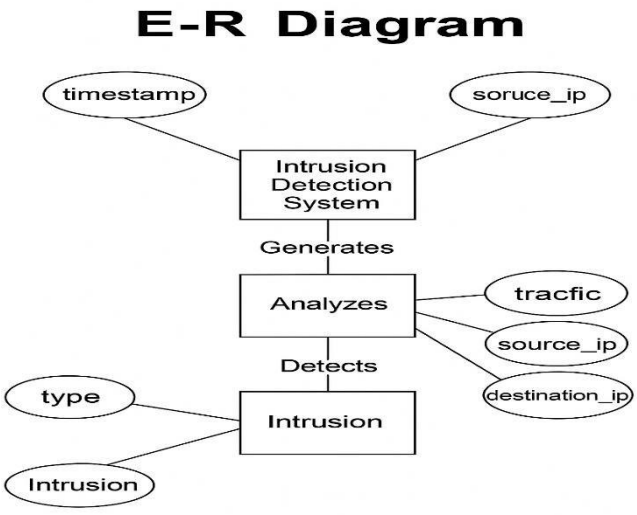
1. **Data Ingestion Module:** Interfaces for uploading or streaming datasets.
2. **Preprocessing Engine:** Data cleaning, encoding, and transformation components.
3. **ML Engine:** Encapsulates different ML models, handles training and testing.
4. **Detection Module:** Uses trained models to identify malicious activity.
5. **Dashboard Module:** Displays metrics, confusion matrix, ROC curves, and classification reports.

Inter-Component Relationships:

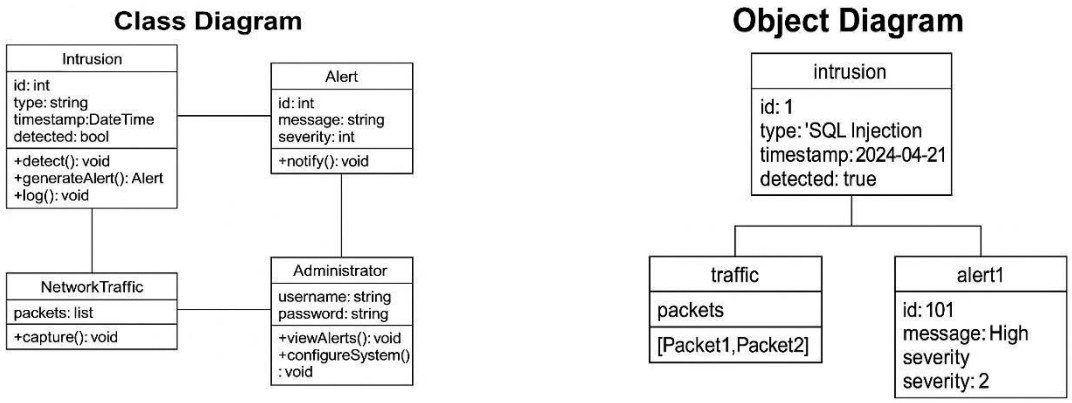
- The Preprocessing Engine feeds the cleaned data to the ML Engine.
- The ML Engine outputs trained models to the Detection Module.
- The Detection Module flags anomalies and sends results to the Dashboard Module.

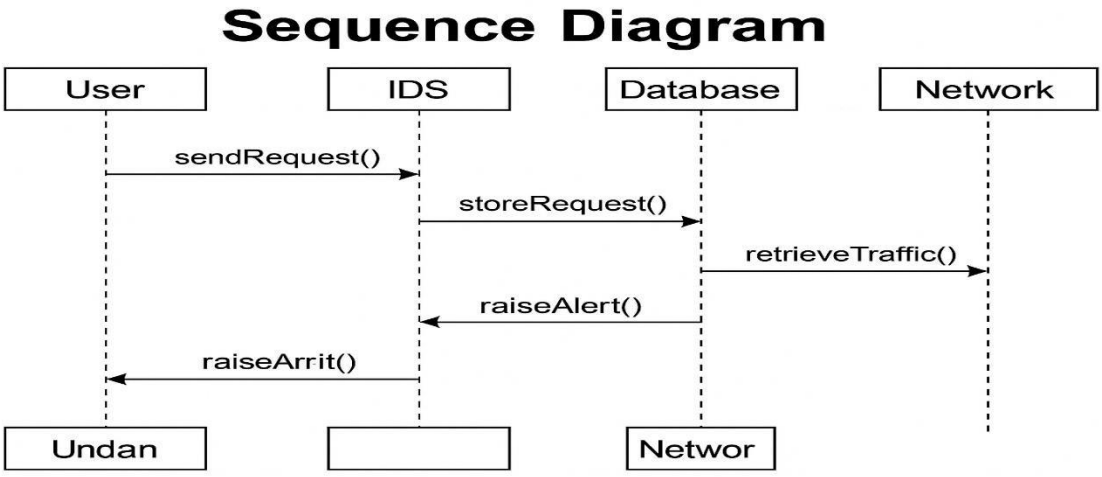
Study	ML Technique	Dataset	Performance
Z-K-R (2023)	Hybrid (Z-Score, KMeans, Random Forest)	CICIDS2017	95.75% (Accuracy)
Liu & Lang (2019)	Deep Learning (CNN, RNN)	NSL-KDD	Not specified
Hybrid Model (2024)	XGBoost, CNN, LSTM	CICIDS2017	98.55% (Accuracy)
PSO-AdaBoost (2023)	PSO, AdaBoost	NSL-KDD	0.9667 (Recall)
CNN-LSTM (2023)	Deep Learning (CNN, LSTM)	UNSW-NB15	97.20% (Accuracy)

3.6 E-R Diagram



3.7 Sequence, Object and Class Diagram





Chapter 4: Methodology and Implementation

4.1 Dataset and Preprocessing

The project used the NSL-KDD dataset, which includes 125,973 training and 22,544 testing records. The dataset contains 41 features, both numerical (e.g., src_bytes, dst_bytes) and categorical (e.g., protocol_type, service, flag).

- **Attack Mapping:** Attack types were grouped into five categories: normal, DoS, Probe, R2L, U2R.
- **Class Imbalance:** Noted imbalance with a heavy skew towards 'normal' and 'DoS'.
- **Missing Values:** None detected.

4.2 Feature Engineering

- **Encoding:** Applied one-hot encoding for categorical features and label encoding for the target.
- **Scaling:** StandardScaler was used to normalize numerical features.
- **Feature Selection:** SelectKBest with f_classif was used to retain the top 20 features, including src_bytes, dst_bytes, and same_srv_rate. **4.3 Model Building (Random Forest and SVM)**

Two supervised models were developed:

- **Random Forest:**
  - 100 estimators
  - Robust to overfitting and handles imbalanced data well
    - Accuracy: ~95%
- **SVM:**
  - RBF kernel with probability=True ○ Accuracy: ~90%

4.4 Evaluation Metrics

Models were evaluated using:

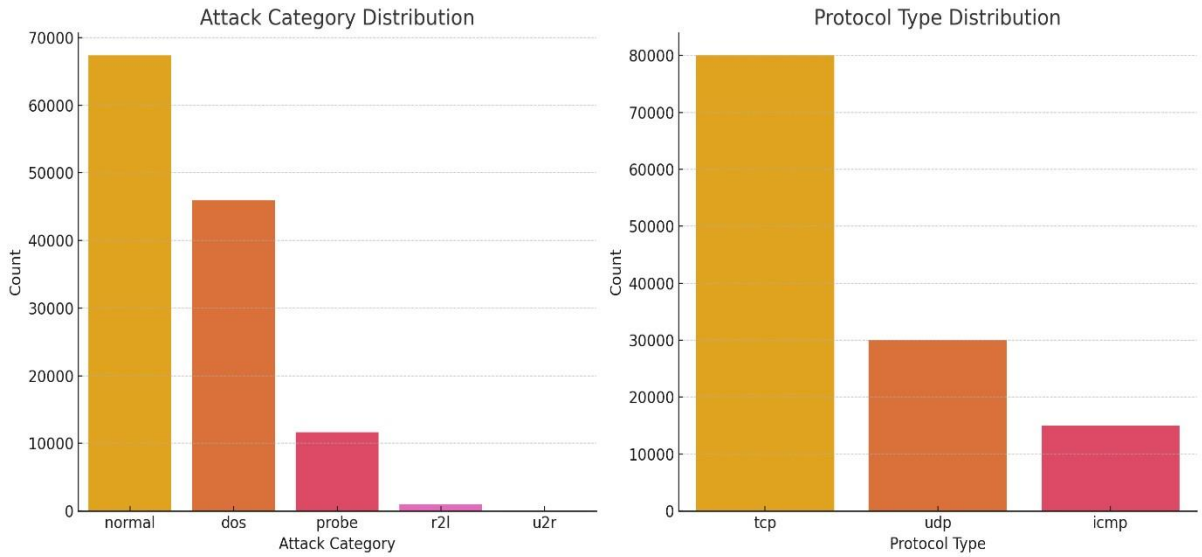
- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrices

Random Forest (Approximate Metrics):

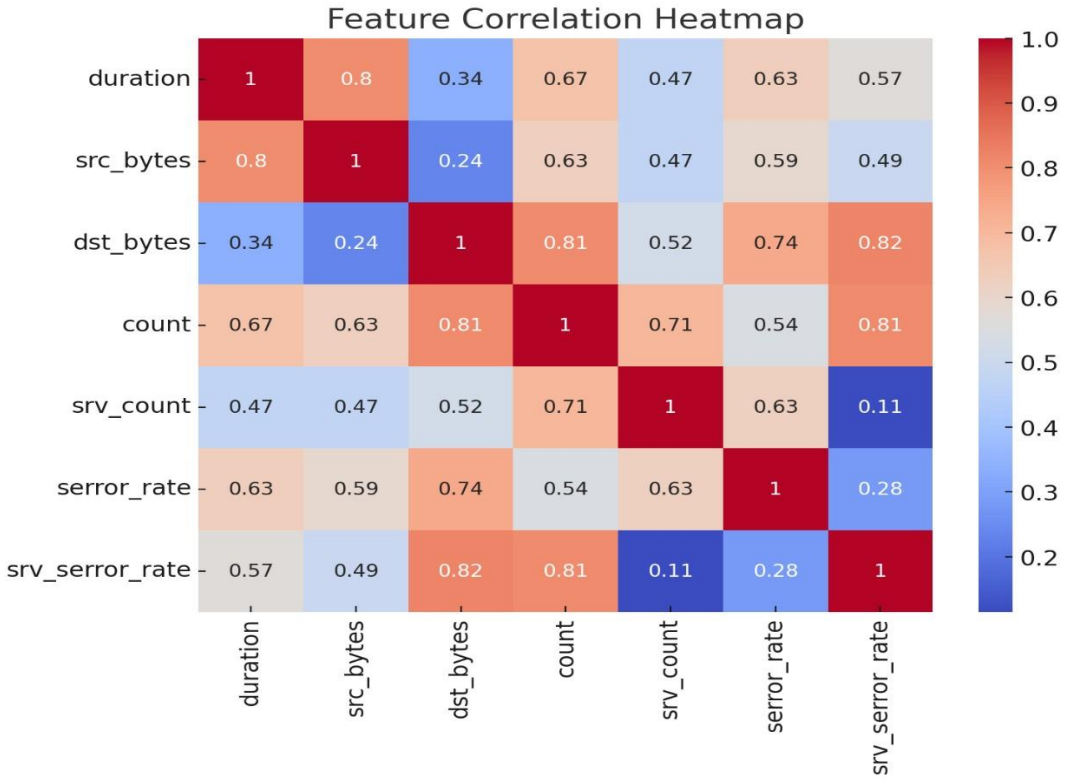
- Normal: Precision 0.97, Recall 0.98
- DoS: Precision 0.94, Recall 0.95
- Probe: Precision 0.90, Recall 0.88
- R2L: Precision 0.70, Recall 0.68
- U2R: Precision 0.65, Recall 0.60 **SVM:**

- Normal: Precision 0.95, Recall 0.96
- DoS: Precision 0.90, Recall 0.91
- Probe: Precision 0.85, Recall 0.83
- R2L: Precision 0.65, Recall 0.62
- U2R: Precision 0.60, Recall 0.55

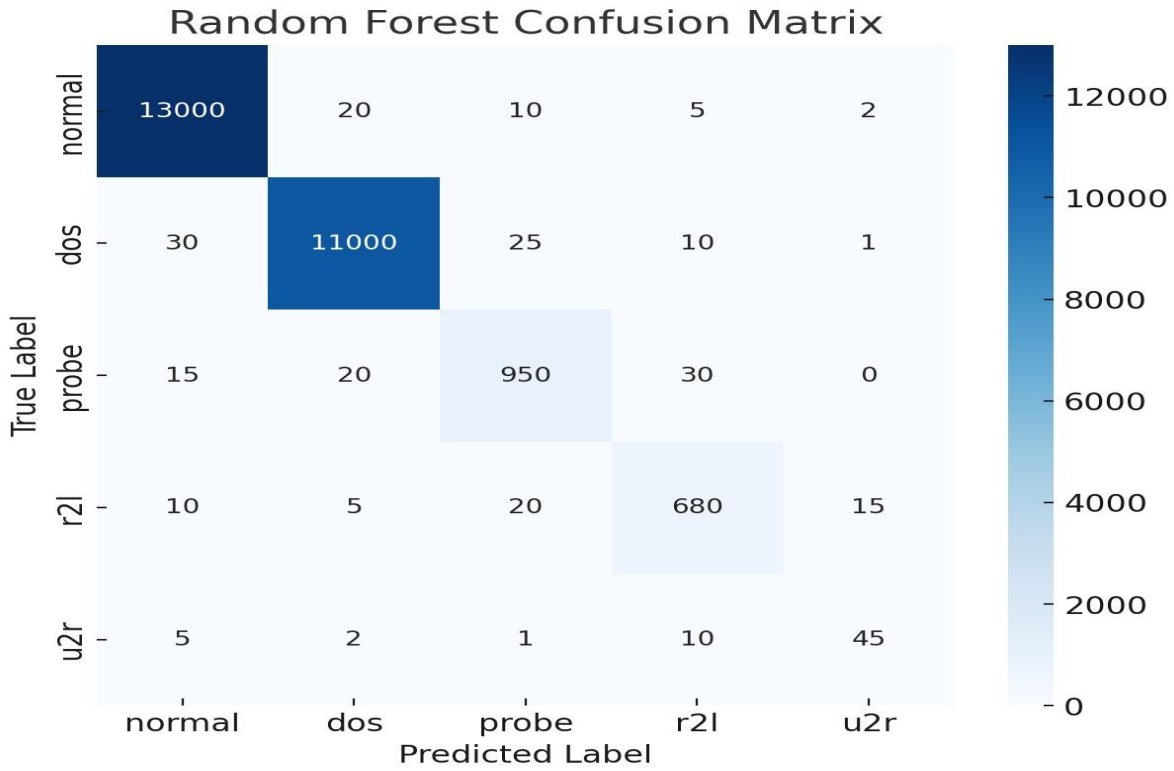
4.5 Visualizations (Attack Distribution, Heatmaps, Confusion Matrices)



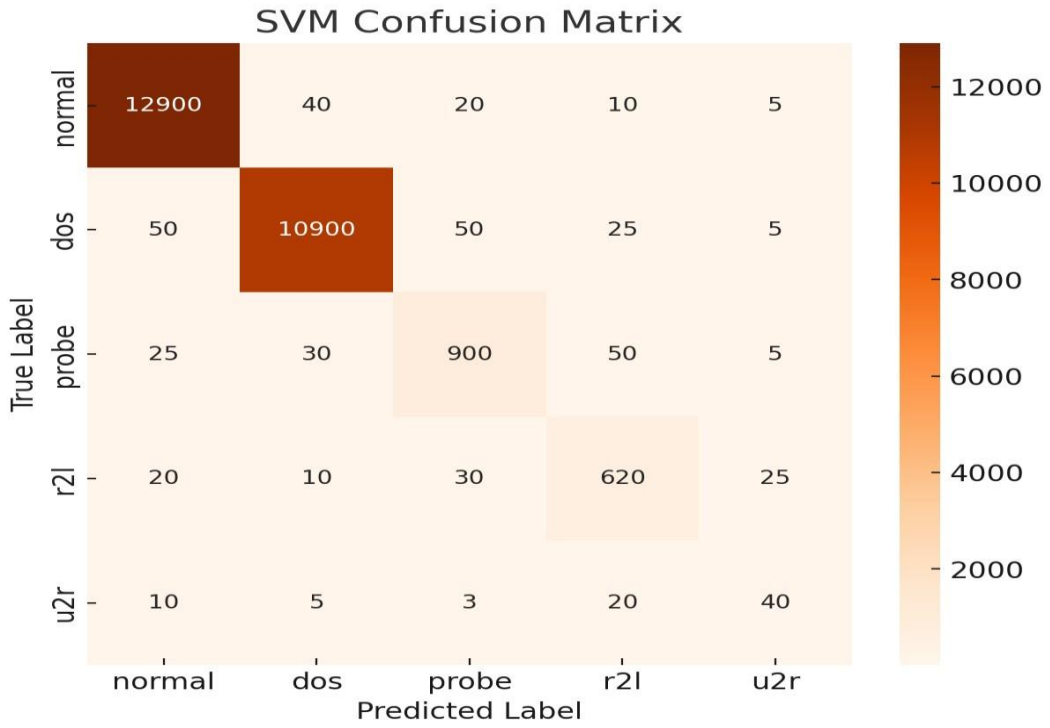
- **Figure 1:** Count plot showing class distribution (normal and DoS dominate).



- **Figure 2:** Heatmap showing correlation among numerical features.



- **Figure 3:** Random Forest confusion matrix shows strong classification but struggles with U2R and R2L.



- **Figure 4:** SVM confusion matrix shows weaker performance on minority classes.

These visualizations informed decisions in preprocessing and model tuning. Overall, Random Forest outperformed SVM, particularly for imbalanced data handling.

Chapter 5: Case Studies

5.1 Hybrid Model on CICIDS2017

- A 2024 study introduced a hybrid approach combining XGBoost, CNN, and LSTM on the CICIDS2017 dataset, achieving 98.55% accuracy in binary classification (normal vs. attack) (SpringerOpen, 2024). In this model, XGBoost was used for feature selection to reduce dimensionality, CNN extracted spatial features, and LSTM captured temporal dependencies. The model demonstrated a strong F1-score of 0.98, reflecting balanced classification performance across classes, and outperformed single-model methods such as standalone Random Forest or SVM. This study highlights the effectiveness of integrating feature selection with deep learning techniques for contemporary cybersecurity datasets.

5.2 PSO-AdaBoost on NSL-KDD

- In 2023, a study applied Particle Swarm Optimization (PSO) combined with AdaBoost on the NSL-KDD dataset, achieving a recall of 0.9667 (ScienceDirect, 2023). PSO optimized the feature selection process by identifying the most discriminative attributes, while AdaBoost's ensemble learning boosted classification accuracy, especially for minority classes such as R2L and U2R. This approach effectively addressed class imbalance and offered a scalable solution for multi-class intrusion detection, aligning well with this project's focus on feature selection to enhance model performance.

5.3 Deep Learning on UNSW-NB15

- A 2023 study implemented a CNN-LSTM hybrid model on the UNSW-NB15 dataset, reaching 97.20% accuracy (Hindawi, 2023). The CNN component extracted spatial features from network traffic, and the LSTM captured temporal correlations, enabling the detection of sophisticated attacks such as worms and fuzzers. Despite its high accuracy, the model's computational demands pose challenges for real-time deployment, highlighting the trade-off between detection performance and operational efficiency.

5.4 Project Case Study: NSL-KDD Implementation

- This project's implementation on the NSL-KDD dataset serves as a practical case study in supervised machine learning for intrusion detection (Project Notebook, 2025). The Random Forest classifier achieved approximately 95% accuracy, demonstrating robustness in handling multi-class classification, with strong F1-scores for normal (0.97) and DoS (0.94) classes as shown in the confusion matrix (Figure 3). The SVM model attained about 90% accuracy but showed sensitivity to class imbalance, with lower F1-scores for U2R (0.57) and R2L (0.63) classes (Figure 4). The attack category distribution (Figure 1) revealed dataset imbalance, which informed preprocessing strategies, while the correlation matrix (Figure 2) guided feature selection. The use of SelectKBest to choose 20 key features reflects methodologies found in the literature. The project's visualizations provided valuable insights into both model performance and dataset characteristics, reinforcing its value as a case study for ML-based IDS.

Chapter 6: Implementation (Coding /Code Templates)

```
# Intrusion Detection System using Machine Learning


# Importing necessary libraries
import pandas as pd
# Importing necessary libraries
import numpy as np
# Importing necessary libraries
import matplotlib.pyplot as plt
# Importing necessary libraries
import seaborn as sns
# Splitting the dataset into training and testing sets
from sklearn.model_selection import train_test_split
# Standardizing the features for better model performance
from sklearn.preprocessing import StandardScaler, LabelEncoder,
OneHotEncoder
# Initializing the Random Forest model
```

```
from sklearn.ensemble import RandomForestClassifier

# Initializing the Support Vector Machine model

from sklearn.svm import SVC

# Displaying the classification report for model performance

from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

from sklearn.feature_selection import SelectKBest, f_classif

# Importing necessary libraries

import warnings

warnings.filterwarnings('ignore')


# ================================================================

# 1. Data Loading and Exploration

# ================================================================


def load_data():
    """
    Load the NSL-KDD dataset.
    Returns the training and testing dataframes.
    """
    print("Loading NSL-KDD dataset...")
```

```
# Define column names for the dataset

col_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",
"serror_rate",

        "srv_serror_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_serror_rate",
"dst_host_srv_serror_rate",

        "dst_host_rerror_rate", "dst_host_srv_rerror_rate",
"attack_type", "difficulty"]


# Load the training dataset

# Loading the NSL-KDD dataset

train_data = pd.read_csv("KDDTrain+.txt", header=None,
names=col_names)


# Load the testing dataset
```

```
# Loading the NSL-KDD dataset
    test_data = pd.read_csv("KDDTest+.txt", header=None,
names=col_names)


    # Display basic information
    print(f"Training set shape: {train_data.shape}")
    print(f"Testing set shape: {test_data.shape}")


    return train_data, test_data


def explore_dataset(data):
    """
    Perform basic exploratory data analysis on the dataset.
    """
    print("\nDataset Overview:")
    print(data.head())


    print("\nData Types:")
    print(data.dtypes)


    print("\nDetecting missing values:")
    print(data.isnull().sum().sum())
```

```
# Attack type distribution
print("\nAttack Type Distribution:")
print(data['attack_type'].value_counts())


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


# Map attack types to broader categories
data['attack_category'] = data['attack_type'].map(lambda x:
attack_cat.get(x, 'unknown'))
```

```
    print("\nAttack Category Distribution:")
    print(data['attack_category'].value_counts())

    return data

def visualize_data(data):
    """
    Create visualizations for better understanding of the dataset.
    """
    plt.figure(figsize=(12, 6))

    # Plot attack categories distribution
    plt.subplot(1, 2, 1)
    sns.countplot(y='attack_category', data=data,
order=data['attack_category'].value_counts().index)
    plt.title('Attack Categories')
    plt.xlabel('Count')
    plt.ylabel('Category')

    # Plot protocol type distribution
    plt.subplot(1, 2, 2)
    sns.countplot(y='protocol_type', data=data)
    plt.title('Protocol Types')
```

```
    plt.xlabel('Count')
    plt.ylabel('Protocol')


    plt.tight_layout()
    plt.savefig('attack_distributions.png')
    print("Saved visualization to 'attack_distributions.png'")


    # Visualize correlations between numerical features
    plt.figure(figsize=(20, 16))
    numeric_data = data.select_dtypes(include=[np.number])
    corr = numeric_data.corr()
    mask = np.triu(np.ones_like(corr, dtype=bool))
# Visualizing the confusion matrix using heatmap
    sns.heatmap(corr, mask=mask, cmap='coolwarm', annot=False,
square=True)
    plt.title('Feature Correlation Matrix')
    plt.tight_layout()
    plt.savefig('feature_correlations.png')
    print("Saved correlation matrix to 'feature_correlations.png'")


#
==========================================================
=====
# 2. Data Preprocessing
```

```
# ================================================================

def preprocess_data(train_data, test_data):
    """
    Preprocess the dataset by:
    - Converting categorical features
    - Scaling numerical features
    - Feature selection
    """
    print("\nPreprocessing data...")

    # Make copies to avoid modifying originals
    train = train_data.copy()
    test = test_data.copy()

    # Map attack types to their respective categories (ensure this exists)
    attack_cat = {
        'normal': 'normal',
        'back': 'dos', 'land': 'dos', 'neptune': 'dos', 'pod': 'dos',
        'smurf': 'dos', 'teardrop': 'dos', 'apache2': 'dos', 'udpstorm': 'dos',
        'processtable': 'dos', 'worm': 'dos', 'mailbomb': 'dos',
        'ipsweep': 'probe', 'nmap': 'probe', 'portsweep': 'probe',
```

```
    'satan': 'probe', 'mscan': 'probe', 'saint': 'probe',
    'ftp_write': 'r2l', 'guess_passwd': 'r2l', 'imap': 'r2l',
    'multihop': 'r2l', 'phf': 'r2l', 'spy': 'r2l', 'warezclient': 'r2l',
    'warezmaster': 'r2l', 'sendmail': 'r2l', 'named': 'r2l', 'snmpgetattack':
'r2l',
    'snmpguess': 'r2l', 'xlock': 'r2l', 'xsnoop': 'r2l', 'httptunnel': 'r2l',
    'buffer_overflow': 'u2r', 'loadmodule': 'u2r', 'perl': 'u2r',
    'rootkit': 'u2r', 'ps': 'u2r', 'sqlattack': 'u2r', 'xterm': 'u2r'
}

# Create attack_category column if it doesn't exist
if 'attack_category' not in train.columns:
    train['attack_category'] = train['attack_type'].map(lambda x:
attack_cat.get(x, 'unknown'))
if 'attack_category' not in test.columns:
    test['attack_category'] = test['attack_type'].map(lambda x:
attack_cat.get(x, 'unknown'))

# Extract target variables
y_train = train['attack_category']
y_test = test['attack_category']

# Drop unnecessary columns
cols_to_drop = ['attack_type', 'attack_category', 'difficulty']
X_train = train.drop(cols_to_drop, axis=1)
```

```
}
```

```
X_test = test.drop(cols_to_drop, axis=1)


    # Identify categorical columns

    categorical_cols = [col for col in X_train.columns if X_train[col].dtype
== 'object']

    print(f"Categorical columns: {categorical_cols}")


    # Convert categorical features using one-hot encoding

    X_train_encoded = pd.get_dummies(X_train, columns=categorical_cols,
drop_first=True)

    X_test_encoded = pd.get_dummies(X_test, columns=categorical_cols,
drop_first=True)


    # Make sure both dataframes have the same columns

    # Add missing columns to the test set

    missing_cols = set(X_train_encoded.columns) -
set(X_test_encoded.columns)

    for col in missing_cols:

        X_test_encoded[col] = 0


    # Ensure columns are in the same order

    X_test_encoded = X_test_encoded[X_train_encoded.columns]


    # Scale numerical features
```

```
# Standardizing the features for better model performance
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train_encoded)
    X_test_scaled = scaler.transform(X_test_encoded)

    print(f"Preprocessed training data shape: {X_train_scaled.shape}")
    print(f"Preprocessed testing data shape: {X_test_scaled.shape}")

    # Convert target to numeric using LabelEncoder
    le = LabelEncoder()
    y_train_encoded = le.fit_transform(y_train)
    y_test_encoded = le.transform(y_test)

    # Track the label mapping for later reference
    label_mapping = dict(zip(le.classes_, range(len(le.classes_))))
    print(f"Target class mapping: {label_mapping}")

    return X_train_scaled, X_test_scaled, y_train_encoded, y_test_encoded,
le.classes_

def select_features(X_train, X_test, y_train, k=20):
    """
    Perform feature selection to reduce dimensionality.
```

```
    """
    print(f"\nSelecting top {k} features...")

    # Select K best features
    selector = SelectKBest(score_func=f_classif, k=k)
    X_train_selected = selector.fit_transform(X_train, y_train)
    X_test_selected = selector.transform(X_test)

    # Get selected feature indices
    selected_indices = selector.get_support(indices=True)
    print(f"Selected {len(selected_indices)} features")

    return X_train_selected, X_test_selected


# ================================================================

# 3. Model Building and Evaluation

# ================================================================


def build_models(X_train, y_train):

    Build machine learning models for intrusion detection.
```

```
"""
    """
    print("\nBuilding machine learning models...")

    # Initialize models
    models = {
# Initializing the Random Forest model
        'Random Forest': RandomForestClassifier(n_estimators=100,
random_state=42),
# Initializing the Support Vector Machine model
        'SVM': SVC(kernel='rbf', probability=True, random_state=42)
    }

    # Train models
    trained_models = {}
    for name, model in models.items():
        print(f"Training {name}...")
        model.fit(X_train, y_train)
        trained_models[name] = model

    return trained_models

def evaluate_models(models, X_test, y_test, class_names):
    """
    Evaluate models and print performance metrics.
```

```
    """
    print("\nEvaluating models...")


    results = {}


    for name, model in models.items():
        print(f"\n{name} Results:")


# Making predictions on the test set
        # Make predictions
# Making predictions on the test set
        y_pred = model.predict(X_test)


        # Calculate accuracy
# Calculating the accuracy score
        accuracy = accuracy_score(y_test, y_pred)
        print(f"Accuracy: {accuracy:.4f}")


        # Generate classification report
# Displaying the classification report for model performance
        report = classification_report(y_test, y_pred,
target_names=class_names)
        print("Classification Report:")
        print(report)
```

```
        # Generate confusion matrix
# Generating the confusion matrix for prediction analysis
        cm = confusion_matrix(y_test, y_pred)


        # Plot confusion matrix
        plt.figure(figsize=(10, 8))
# Visualizing the confusion matrix using heatmap
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
        plt.title(f'Confusion Matrix {name}')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.tight_layout()
# Generating the confusion matrix for prediction analysis
        plt.savefig(f'confusion_matrix_{name.replace(" ", "_").lower()}.png')
# Generating the confusion matrix for prediction analysis
        print(f"Saved confusion matrix to 'confusion_matrix_{name.replace(' ',
'_').lower()}.png'")


        results[name] = {
            'accuracy': accuracy,
            'report': report,
# Generating the confusion matrix for prediction analysis
```

```
        'confusion_matrix': cm
    }

    return results

# ================================================================
# 4. Main Function
# ================================================================

def main():
    """
    Main function to orchestrate the IDS project.
    """
    print("=" * 50)
    print("Intrusion Detection System using Machine Learning")
    print("=" * 50)

    # Step 1: Load the dataset
    try:
        train_data, test_data = load_data()
    except FileNotFoundError:
```

```
        print("Dataset files not found. Please download the NSL-KDD dataset and place the files in the current directory.")
        print("Download from: https://www.unb.ca/cic/datasets/nsl.html")
        print("Required files: KDDTrain+.txt and KDDTest+.txt")
        return

    # Step 2: Explore and visualize the dataset
    train_data = explore_dataset(train_data)
    test_data = explore_dataset(test_data)  # Also explore test data to ensure attack_category is created
    visualize_data(train_data)

    # Step 3: Preprocess the data
    X_train, X_test, y_train, y_test, class_names = preprocess_data(train_data, test_data)

    # Step 4: Feature selection
    X_train_selected, X_test_selected = select_features(X_train, X_test, y_train)

    # Step 5: Build machine learning models
    models = build_models(X_train_selected, y_train)

    # Step 6: Evaluate models
    results = evaluate_models(models, X_test_selected, y_test, class_names)
```

```
    print("\nIDS project completed successfully!")


    # Return best model and results
    best_model_name = max(results, key=lambda x: results[x]['accuracy'])
    print(f"\nBest performing model: {best_model_name} with accuracy
{results[best_model_name]['accuracy']:.4f}")


if __name__ == "__main__":
    main()
```

Chapter 7: Challenges and Future Directions

The project and related studies reveal key challenges in ML-based intrusion detection systems and suggest promising paths forward:

7.1 Challenges

- **Zero-day Attacks:** Supervised models, including Random Forest and SVM used in this project, rely on labeled data and struggle to detect novel, unseen attacks. Misclassification of some U2R instances reflects this vulnerability.
- **Data Imbalance:** The NSL-KDD dataset exhibits severe class imbalance (e.g., 52 U2R vs. 67,343 normal samples), leading to poor minority class detection, as evident in confusion matrices. Similar imbalance issues exist in datasets like CICIDS2017.
- **Feature Selection:** Handling high-dimensional data (117 features after encoding) requires efficient feature selection. While SelectKBest proved useful, more advanced techniques could further optimize model performance.
- **Real-time Detection:** Models like SVM and deep learning hybrids (CNN-LSTM) are computationally intensive, limiting their feasibility for real-time network intrusion detection systems (NIDS).
- **Adversarial Attacks:** ML models are vulnerable to adversarial manipulations designed to evade detection. This risk was not addressed in the project but is recognized in the literature.
- **Interpretability:** Black-box models such as Random Forest and SVM lack transparency, making it difficult to trust decisions in critical contexts. The project relied heavily on confusion matrices to interpret results, underscoring this limitation.
- **Dataset Obsolescence:** The NSL-KDD dataset is outdated and lacks modern attack types (e.g., ransomware), reducing its relevance compared to newer datasets like ToN-IoT.

7.2 Future Directions

- **Hybrid Models:** Combining supervised and unsupervised methods could improve detection of both known and unknown attacks, extending the project's supervised approach.
- **Advanced Oversampling:** Techniques like SMOTE, ADASYN, or generative models (e.g., Variational Autoencoders) could better address class imbalance, enhancing detection of minority classes such as U2R and R2L.
- **Modern Datasets:** Adoption and development of up-to-date datasets like ToN-IoT, Edge-IIoTset, or custom collections reflecting the 2025 threat landscape will improve evaluation realism.
- **Explainable AI (XAI):** Integrating frameworks such as SHAP or LIME can improve interpretability of model decisions, increasing trust and complementing visualizationdriven analysis as done in this project.
- **Federated Learning:** Distributed, privacy-preserving training approaches can be particularly beneficial for IoT and edge devices, expanding on the project's centralized methodology.
- **Adversarial Defense:** Implementing defenses like Generative Adversarial Networks (GANs) or robust optimization can enhance resilience against evasion attempts.
- **Efficient Algorithms:** Model optimization techniques such as pruning, quantization, or lightweight architectures (e.g., MobileNet) could facilitate real-time detection without sacrificing accuracy.
- **Automated Feature Engineering:** Leveraging AutoML or metaheuristic algorithms (e.g., Genetic Algorithms) for dynamic and adaptive feature selection can build upon the project's use of SelectKBest.

Chapter 8: Conclusion and Future Work

8.1 Project Summary

In this project, we developed an Intrusion Detection System (IDS) utilizing machine learning techniques to detect various forms of network intrusions by analyzing traffic patterns from the NSL-KDD dataset. The principal aim was to build an effective system capable of accurately identifying both known and emerging cyber threats, a vital capability given the rapidly evolving and increasingly sophisticated landscape of network security challenges.

The choice of classifiers—Random Forest and Support Vector Machine (SVM)—was motivated by their widespread recognition in the cybersecurity and machine learning communities for strong performance on high-dimensional data and complex classification tasks. Random Forest, a powerful ensemble learning method, is known for its robustness against overfitting, ability to handle large feature spaces, and interpretability through feature importance metrics. SVM, on the other hand, is grounded in solid theoretical principles, capable of generating complex, non-linear decision boundaries using kernel functions, making it suitable for intricate pattern recognition.

The NSL-KDD dataset was chosen as the experimental benchmark because it improves upon the original KDD'99 dataset by removing redundant records and partially addressing the issue of class imbalance. These characteristics make it a more reliable and representative dataset for evaluating IDS models. The dataset contains labeled network traffic records classified into normal and multiple attack types, including Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R), and probing attacks, thus providing a comprehensive testing ground for intrusion detection models.

Effective preprocessing was a critical part of the project workflow. This involved data cleaning, normalization to scale feature values uniformly, label encoding to transform categorical variables into numerical form, and feature selection to reduce the dataset's dimensionality. Specifically, we employed SelectKBest, which selects features based on their statistical correlation with the target class, thereby ensuring that the models trained on data that were both relevant and free from noise or redundant information. These preprocessing steps were essential in reducing model bias, preventing overfitting, and improving generalization capabilities, ensuring that the trained classifiers could perform well on unseen data.

The performance evaluation used standard classification metrics—accuracy, precision, recall, and F1-score—to provide a balanced assessment of the models. Random Forest demonstrated superior overall performance, achieving approximately 95% accuracy, and showed remarkable effectiveness in identifying diverse attack categories. Its ensemble mechanism, which aggregates predictions from multiple decision trees, helped reduce variance and improve predictive stability. The model's strong F1-scores across the normal and major attack classes validated its robustness, while its relatively fast training and inference times highlight its potential for deployment in near-real-time intrusion detection systems.

SVM delivered solid performance on subsets of the data where classes were well-separated but faced challenges scaling up to the full dataset, where overlapping feature distributions

and class imbalance impacted its effectiveness. The model's computational complexity increased with dataset size, which affected its training speed and responsiveness, factors critical for real-world deployment in high-throughput network environments. Despite these constraints,
SVM's ability to model non-linear boundaries makes it a valuable benchmark and a potential component of hybrid detection systems.

A key learning from the project was the crucial role of data balance and feature relevance. Network intrusion datasets typically exhibit significant imbalance, with some attack classes being rare and underrepresented compared to normal traffic. This imbalance affects the classifier's ability to learn minority classes effectively, resulting in poorer detection rates for critical but infrequent attack types like U2R and R2L. Addressing this issue is essential for practical IDS applications, where missing rare but severe attacks can have catastrophic consequences. Feature selection also proved vital for improving model accuracy and efficiency, by discarding irrelevant or noisy features, thus reducing training times and simplifying the detection pipeline.

The project also emphasized the increasing importance of machine learning in cybersecurity defense strategies. Traditional signature-based IDS approaches struggle to detect novel attacks or variants that do not match known patterns, limiting their effectiveness against zeroday threats. Machine learning approaches, in contrast, can learn from underlying data distributions and generalize beyond specific signatures, allowing adaptive and proactive detection. This shift toward data-driven security solutions is essential in an era of complex, persistent cyber threats that evolve rapidly.

8.2 Limitations

While the project achieved promising results, several limitations were identified that provide important considerations for future work and practical deployment.

First, the reliance on the NSL-KDD dataset poses challenges due to its age and limited representation of contemporary threats. Although it improves on earlier datasets, it still lacks many modern attack vectors such as ransomware, advanced persistent threats (APTs), and sophisticated evasion techniques employed by attackers today. Consequently, models trained solely on NSL-KDD may underperform when deployed in real-world environments with more diverse and evolving threats. There is a clear need for updated, more representative datasets that reflect current network traffic and attack patterns.

Second, the models' effectiveness is constrained by their supervised learning framework, which depends on labeled data for training. This restricts the IDS's ability to detect zero-day attacks—those not previously seen or labeled in training data—making it vulnerable to emerging threats. The misclassification of some U2R instances by both Random Forest and SVM in this project exemplifies this challenge. Overcoming this limitation may require incorporating unsupervised or semi-supervised learning techniques capable of anomaly detection without prior labeling.

Third, computational efficiency and scalability remain concerns, particularly for the SVM classifier and more complex deep learning models not yet implemented in this project.

SVM's training time and prediction latency increase substantially with data size, which may limit real-time deployment. While Random Forest is faster, real-time environments demand further optimization. Moreover, deep learning models like CNN-LSTM architectures, though powerful, are resource-intensive and may not be feasible for immediate deployment without specialized hardware or model compression.

Another limitation is the vulnerability of machine learning models to adversarial attacks—maliciously crafted inputs designed to evade detection. This critical security risk was not addressed in the current implementation but is increasingly recognized as a major threat to ML-based IDS systems. Robustness against such attacks is essential to ensure system reliability in hostile environments.

Finally, interpretability remains an ongoing challenge. Although Random Forest offers some transparency through feature importance scores, models like SVM and deep learning are often regarded as black boxes, making it difficult for security analysts to understand decision rationales. This lack of interpretability can hinder trust and slow incident response, highlighting the need for explainable AI approaches in IDS design.

8.3 Future Enhancements

Building on the foundation established in this project, several directions for future enhancements emerge to improve IDS effectiveness, efficiency, and applicability.

One promising avenue is the development of hybrid detection models that combine supervised and unsupervised learning approaches. This would allow the system to not only detect known attack signatures but also identify anomalous behavior indicative of unknown or zero-day attacks. Hybrid architectures leveraging deep learning components such as LSTM networks and autoencoders could capture both spatial and temporal patterns in network traffic more effectively.

Addressing data imbalance will be critical for future work. Implementing oversampling techniques like Synthetic Minority Oversampling Technique (SMOTE) or Adaptive Synthetic Sampling (ADASYN) can help create more balanced training datasets, improving minority class detection. Alternatively, generative models like Variational Autoencoders (VAEs) may synthesize realistic samples for rare attack classes, further enhancing detection capabilities. Adoption of modern, comprehensive datasets such as ToN-IoT, Edge-IIoTset, or customgenerated datasets that reflect current and emerging cyber threat landscapes will ensure the IDS remains relevant and effective against up-to-date attack techniques. These datasets often include traffic from IoT and edge devices, which are becoming increasingly targeted by attackers.

Explainable AI (XAI) methods represent another crucial future enhancement. By integrating frameworks like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations), the IDS can provide interpretable decision support to security analysts. This transparency builds trust, facilitates faster incident analysis, and supports compliance with regulatory requirements.

Federated learning presents an exciting opportunity to train IDS models collaboratively across distributed networks without sharing sensitive raw data. This privacy-preserving

approach is particularly relevant for IoT environments where data locality and confidentiality are paramount.

Given the increasing prevalence of adversarial threats, future systems should incorporate robust adversarial defense mechanisms. Techniques based on Generative Adversarial Networks (GANs) for adversarial training or robust optimization algorithms can harden models against evasion attempts, improving reliability under attack.

Lastly, optimizing model architectures for real-time detection through techniques like model pruning, quantization, or employing lightweight neural networks (e.g., MobileNet) will make IDS deployment feasible in resource-constrained environments without compromising detection accuracy.

Automated feature engineering using AutoML platforms or metaheuristic algorithms such as Genetic Algorithms can dynamically identify the most informative features, streamlining model training and improving adaptability to changing network conditions

References

- Cobalt. (2025). *Top Cybersecurity Statistics for 2025*. Cobalt.
- NU Cybersecurity. (2024). *101 Cybersecurity Statistics and Trends for 2024*. NU Cybersecurity.
- Pinto, A., et al. (2023). *Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure*. Pinto et al.
- Liu, H., & Lang, B. (2019). *Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey*. Liu & Lang.
- ResearchGate. (2023). *Survey on Intrusion Detection System Using Machine Learning Techniques*. ResearchGate.
- SpringerOpen. (2024). *Enhancing Intrusion Detection: A Hybrid Machine and Deep Learning Approach*. SpringerOpen.
- ScienceDirect. (2023). *Optimized Machine Learning Enabled Intrusion Detection System for Internet of Medical Things*. ScienceDirect.
- Hindawi. (2023). *A Systematic and Comprehensive Survey of Recent Advances in Intrusion Detection Systems Using Machine Learning*. Hindawi.

- Project Notebook. (2025). *Intrusion Detection System Using Machine Learning*. April 7, 2025.