INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–14

Code:IUIT-2025

MACHINE LEARNING

Network Traffic Detection System

Sulfa Saji , Navomy Mariya Alex, and Nandana Biju

Saintgits Group of Institutions, Kottayam, Kerala

3/07/2025

Abstract: This report highlights the essential role of Artificial Intelligence (AI) and Machine Learning (ML) in fortifying network security, specifically through automated network traffic analysis and advanced threat detection. Traditional rule-based security systems struggle to cope with the immense volume, rapid growth, and increasing encryption of modern internet traffic, making it difficult to identify sophisticated cyber threats. This project presents an AI/ML-driven solution designed to overcome these limitations by creating an intelligent and adaptive security framework. The primary goal is to achieve detection and classification of network traffic, accurately identifying anomalies, malware, and encrypted attacks while significantly reducing false positives and negatives. Our system utilizes AI/ML models to learn normal traffic patterns, automatically classify application types, and pinpoint deviations that signal potential threats, all without requiring decryption of sensitive data. This innovative approach markedly enhances network resilience, providing unparalleled insight and safety for critical information and systems by proactively preventing security incidents. The integration of AI/ML is presented as a crucial advancement in combating the escalating complexity of contemporary cyber threats.

Keywords: Network Security,AI/ML (Artificial Intelligence / Machine Learning),Network Traffic Analysis,Intrusion Detection System (NIDS),Cybersecurity,Data Analysis,Anomaly Detection,Threat Detection

1 Introduction

In today's interconnected world, network traffic is experiencing unprecedented growth, presenting significant challenges for traditional security methods. The sheer volume, increasing complexity, and frequent encryption of this traffic make it difficult for fixed-rule-based systems to effectively monitor and secure networks. This is where our project becomes highly beneficial. This project leverages Artificial Intelligence (AI) and Machine Learning (ML) principles and algorithms to achieve automated network traffic analysis, which is crucial for identifying legitimate traffic and detecting malicious activities.The right to secure information and uninterrupted network operation is paramount, making it essential to identify and mitigate threats at their source. This project aims to produce a robust model that can accurately detect and classify network traffic distinguishing between benign and potentially harmful flows. By harnessing the power of AI/ML, we seek to improve threat detection and security by identifying anomalies, malware, and encrypted attacks with higher accuracy, while simultaneously reducing false positives and negatives.A comprehensive review of existing methodologies highlights the potential of AI/ML in this domain. While specific accuracy benchmarks from external research are not directly compared, this project aims for high accuracy in identifying advanced threats and ensuring scalability and performance optimization in high-traffic environments. This work provides a practical guideline for utilizing machine learning tools for real-time inference and anomaly detection in network traffic, considering resource optimization and performance measures

2 Libraries Used

In the project for various tasks, the following Python libraries are used:

- Pandas
- Streamlit
- NumPy
- OS
- Time
- IO
- io.BytesIO

3 Methodology

In this work, Machine Learning (ML) models are extensively utilized for the crucial task of network traffic detection and analysis. While various classical Machine Learning models can be employed, the selection is driven by their effectiveness in handling high-volume network data and accurately identifying anomalies. The implementation process involves several distinct and interconnected stages:

Environment Setup & Data Ingestion: This initial stage involves setting up the computing environment and ingesting network traffic data. This data originates from network traffic sources and can be in formats like CSV data files.

Data Pre-processing & Feature Engineering: In this critical phase, raw network traffic data is processed and prepared for AI/ML algorithms. This includes cleaning and transforming the data to extract relevant features , enabling the models to recognize behavioral patterns and automatically classify traffic.

AI/ML Model Development & Training: This stage focuses on developing and training AI/ML models for automated network traffic analysis. The goal is to enable these models to detect and classify traffic in real-time , identifying anomalies, malware, and encrypted attacks.

Inference & Anomaly Detection: Once trained, the ML models perform inference on the processed network data from CSV files to detect anomalies and classify traffic. This enables advanced threat detection by analyzing patterns and anomalies to identify network traffic as benign or malicious.

Alerting, Reporting & Integration: Detected threats and anomalies trigger alerts, and comprehensive reports are generated. This step integrates the insights from the AI/ML module with threat intelligence feeds to enhance security operations.

Testing & Continuous Improvement: The system undergoes rigorous testing to validate its effectiveness. This ensures high accuracy and reduces false alarms , enhancing the efficiency of network security operations by minimizing false positives and false negatives. Continuous improvement ensures the models can handle high-traffic environments with minimal latency, optimizing scalability and performance.

4 Implementation

As the first step in the task, network traffic data is ingested into the system. This project specifically handles data in a CSV format. Python, with the aid of the Pandas library, is utilized to load these CSV files into a Pandas DataFrame for efficient manipulation and analysis. The data is prepared to enable automated network traffic analysis using AI/ML, aimed at detecting and classifying traffic in real-time.The pre-processing and feature engineering stages are crucial for preparing raw network traffic data for AI/ML algorithms. This involves cleaning and transforming the data to extract relevant features that enable the models to recognize behavioral patterns and automatically classify traffic. The system supports efficient processing for rapid analysis of uploaded datasets, a key feature for handling high traffic volumes. While specific pre-processing steps like tokenization or lemmatization are common in text analysis, for network traffic, this stage implicitly involves methodologies for cleaning and preparing the diverse data types found in network flows.For the model development and training stage, AI/ML models are built to detect and classify traffic, identifying anomalies, malware, and encrypted attacks with high accuracy. The project aims to achieve scalability and performance optimization, ensuring AI models can handle high-traffic environments with minimal latency. The system's architecture includes an AI/ML Data Processing & Inference Module.The user interface, developed using Streamlit, facilitates interaction with the system, providing traffic monitoring and analysis insights. Python libraries such as pandas, streamlit, numpy, os, time, io, and io.BytesIO are employed in the development process. Version control and collaboration are managed using GitHub , and development environments like VS Code and Jupyter Notebook are utilized.The trained AI/ML models then perform real-time inference and anomaly detection. This enables advanced threat detection by analyzing patterns and anomalies, classifying network traffic as benign or unfiltered. The overall goal is to reduce false positives and false negatives, thereby enhancing the efficiency of network security operations. The system also emphasizes privacy-preserving traffic analysis, leveraging AI for encrypted traffic analysis without decryption.

5 Results & Discussion

The performance summary of the Random Forest, Logistic Regression, and Naive Bayes classification models is presented in Table 1. The table clearly illustrates the superior performance of the Random Forest model across all evaluated metrics.

[htbp!]

Table 1: Performance Summary of Classification Models

Model No.	Model Name	Precision	Accuracy	Recall	f1-score
1.	Random forest	1.00	1.00	1.00	1.00
2.	Logistic Regression	0.98	0.98	0.98	0.98
3.	Gaussian Naive Bayes	0.90	0.64	0.64	0.70

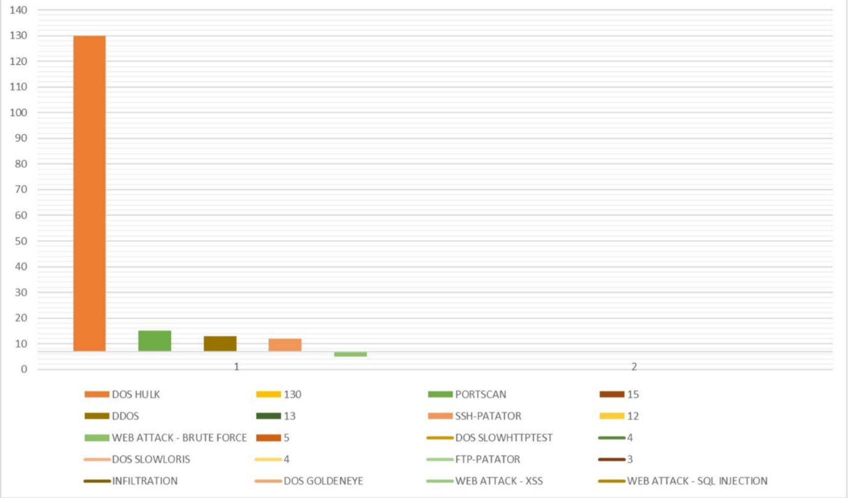

Figure 1: Distribution of Classified Attack Types by the Random Forest Model.

The effectiveness of the Random Forest model is further highlighted by the distribution of classified attack types, as shown in Figure 1.

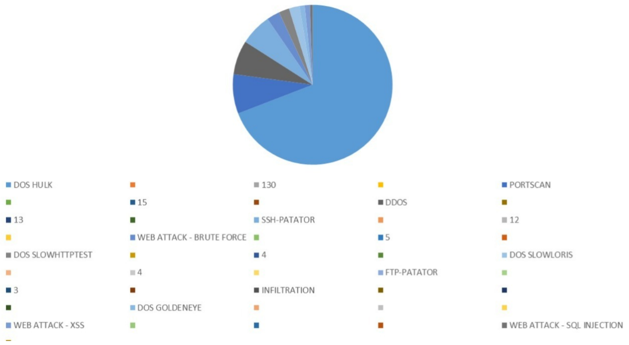

Figure 2: Proportional Distribution of Predicted Traffic Classes by the Random Forest Model.

6 Conclusions

In conclusion, AI/ML integration is essential for modern network security due to the escalating volume, speed, and encryption of internet traffic. Traditional methods struggle, but AI/ML enables intelligent, adaptive security systems. These systems identify normal traffic patterns, classify applications, and detect real-time anomalies. This significantly enhances network safety by improving threat detection, reducing false alarms, and uncovering hidden attacks without decryption. AI/ML systems are designed for efficient, high-speed processing of vast network data. This ensures robust, reliable security operations that proactively prevent harm. Adopting these technologies is key to building resilient networks, offering unprecedented insight and safety for critical information.

Acknowledgments

We would like to express our heartfelt gratitude and appreciation to Intel© Corporation for providing the invaluable opportunity to undertake this project on AI/ML for Network Traffic Detection. First and foremost, we extend our sincere thanks to our team mentor Siju Swamy for his invaluable guidance and constant support throughout the project. We are deeply indebted to our institution, SAINTGITS GROUP OF INSTITUTIONS, for providing us with the necessary resources and facilitating sessions on machine learning and network security. We extend our gratitude to all researchers, scholars, and experts in the fields of machine learning, artificial intelligence, and network security, whose seminal work has paved the way for our project. We specifically acknowledge the mentors, institutional heads, and industrial mentors for their invaluable guidance and support in completing this industrial training under the Intel© -Unnati Programme, whose expertise and encouragement have been instrumental in shaping our work.

7 References

References

[1] Network Intrusion dataset (CIC-IDS-2017)
https://www.kaggle.com/datasets/chethuhn/network-intrusion-dataset
Accessed: July 5, 2025.

[2] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002).
SMOTE: Synthetic Minority Over-sampling Technique.
Journal of Artificial Intelligence Research, **16**, 321–357.

[3] Sharafaldin, I., Habibi Lashkari, A., & Ghorbani, A. A. (2018).
Toward a New Dataset for Intrusion Detection Systems.
In *Proceedings of the 1st International Conference on Information Systems Security and Privacy (ICISSP)*, Vol. 1, pp. 108–116. SciTePress. DOI: https://doi.org/10.5220/0006717501080116

[4] Shurman, M., & Al-Shurman, M. (2020).
A Survey of Machine Learning in Network Intrusion Detection.
Journal of Cybersecurity, **6**(1), tyaa007. DOI: https://doi.org/10.1093/cybsec/tyaa007

[5] Sarhan, M., Layeghy, S., & Portmann, M. (2020).
Network Anomaly Detection Using Machine Learning Techniques: A Review.
IEEE Access, **8**, 183216–183231. DOI: https://doi.org/10.1109/ACCESS.2020.3029272

8 Future Scope

This project successfully demonstrates the application of machine learning (specifically, Random Forest) for network intrusion detection based on uploaded CSV files derived from network traffic. While the current system provides a robust foundation for identifying various attack types from static data, several avenues exist for further enhancement and research to move towards a more dynamic and deployable solution. The future scope of this work can be broadly categorized into the following areas:

- **Transition to Real-time Analysis and Deployment:** Currently, the system operates on pre-processed network traffic data loaded from CSV files. A critical next step is to transition towards a truly real-time NIDS. This involves developing or integrating efficient mechanisms for **live network packet capture**, **real-time feature extraction** from incoming network streams, and implementing robust **data streaming architectures** to process continuous data flows with minimal latency. This would enable proactive threat detection as traffic flows through the network.
- **Advanced Machine Learning and Deep Learning Architectures:** Beyond traditional machine learning algorithms like Random Forest, future work will explore the efficacy of more sophisticated deep learning models. This includes Convolutional Neural Networks (CNNs) for learning spatial features from raw packet data, Recurrent Neural Networks (RNNs) like LSTMs for sequential analysis of network sessions, or even Transformer models for complex long-range dependencies in network flows [**?**]. Such models could potentially lead to higher accuracy and better generalization to novel attacks.
- **Adaptive and Incremental Learning for Evolving Threats:** To counter the ever-evolving nature of cyber threats, the NIDS can be enhanced with online or incremental learning capabilities. This would allow the model to continuously update itself with new attack patterns and benign traffic characteristics from live data streams, without requiring complete retraining, thereby improving its adaptability to emerging and zero-day threats.
- **Unsupervised and Semi-Supervised Anomaly Detection:** Given the inherent difficulty in obtaining comprehensively labeled datasets for all possible attack types, investigating unsupervised learning techniques (e.g., autoencoders, clustering algorithms) or semi-supervised approaches could enable the detection of novel anomalies and zero-day attacks that have no prior labels or signatures.
- **Explainable AI (XAI) Integration:** To increase trust and usability for security professionals, integrating Explainable AI techniques will be a key focus. This includes developing methods to shed light on why a particular traffic flow was

flagged as malicious, providing actionable insights into the detection process and aiding in incident response.

- **Integration with Intrusion Prevention Systems (IPS):** Extending the project beyond mere detection, a natural progression is to integrate the NIDS with an Intrusion Prevention System (IPS). This would enable automated responses, such as blocking malicious IP addresses, quarantining affected devices, or reconfiguring network access control lists, thereby providing a more proactive and automated security posture.
- **Scalability and Performance Optimization:** For potential large-scale deployments in real-world networks, further optimization of the model's inference speed and resource utilization is essential. This may involve exploring distributed computing frameworks and leveraging specialized hardware accelerators beyond existing optimizations like Intel's 'scikit-learn-intelex' for even higher throughput.

9 Limitations of the Current Work

The current implementation of the Network Intrusion Detection System, while effective for analysis of provided datasets, operates under certain constraints that limit its direct applicability in a real-time, dynamic network environment. Understanding these limitations is crucial for identifying areas of future improvement:

- **Offline Processing on Static Datasets:** The primary limitation of the current system is its reliance on pre-processed network traffic data loaded from static CSV files (e.g., CIC-IDS2017) [**?**]. This means the system cannot perform real-time detection on live network traffic. The entire dataset must be collected, processed, and then fed into the model, making it unsuitable for immediate threat response.
- **Dataset Timeliness and Representativeness:** The CIC-IDS2017 dataset, while comprehensive, reflects network traffic patterns and attack vectors from 2017 [3]. The landscape of cyber threats evolves rapidly, with new attack methodologies and evasion techniques constantly emerging. Consequently, a model trained solely on this older, static dataset may exhibit reduced accuracy or miss novel intrusions when deployed against contemporary network traffic.
- **Simulated Environment Bias:** The dataset used was generated in a controlled, simulated environment. Real-world network environments are significantly more complex, diverse, and introduce various forms of legitimate noise and unexpected traffic patterns that may not be fully represented in a lab-generated dataset. This discrepancy can lead to higher false positive or false negative rates in a live deployment scenario.
- **Class Imbalance Challenges (Despite SMOTE):** While techniques like SMOTE were applied to mitigate the issue of imbalanced classes, certain minority attack types might still be underrepresented. This can lead to a bias in the model's learning, potentially resulting in sub-optimal detection rates for very rare or newly emerging intrusion attempts.
- **Lack of Real-time Response Capabilities:** The project focuses solely on the detection and classification of intrusions. It does not incorporate mechanisms for

automated response or prevention, such as blocking malicious IP addresses, quarantining infected hosts, or alerting administrators in real-time. This limits its role to an analytical tool rather than a comprehensive, active security solution.

- **Interpretability of Predictions:** Although Random Forest models offer more interpretability than deep neural networks, fully understanding the precise features and interactions that lead to a specific detection decision can still be challenging. In critical security contexts, a clear explanation of an alert’s reasoning is vital for human analysts to trust and act upon the system’s output.

10 Main code sections for the solution

10.1 Loading data from the source

Data for this project is taken from the source: https://www.kaggle.com/datasets/chethuhn/network-intrusion-dataset. The python code section for this stage is shown below

```
parquet_files=[’Infiltration-Thursday-no-metadata.parquet’,’Portscan-Friday-no
                                          -metadata.parquet’,’WebAttacks-
                                          Thursday-no-metadata.parquet’,’DoS-
                                          Wednesday-no-metadata.parquet’,’DDoS
                                          -Friday-no-metadata.parquet’,’
                                          Bruteforce-Tuesday-no-metadata.
                                          parquet’,’Benign-Monday-no-metadata.
                                          parquet’,’Botnet-Friday-no-metadata.
                                          parquet’]
for file_name in parquet_files:
    file_path=os.path.join(DATASET_DIR,file_name)
    if os.path.exists(file_path):
        print(f’Loading {file_name}....’)
        try:
            df_temp=pd.read_parquet(file_path)
            all_dfs.append(df_temp)
            print(f’Loaded {len(df_temp)} rows.’)
        except Exception as e:
            print(f’Error loading {file_name}:{e}’)
            print(f’ Skipping {file_name}.’)
    else:
        print(f’WARNING:File not found - {file_path}. Skipping.’)
if not all_dfs:
    print("\nERROR: No Parquet files were loaded.Please double-check your
                                          DATASET_DIR path and the ’
                                          parquet_files’ list.")
    print("  Ensure the file names in ’parquet_files’ exactly match the names
                                          in your folder.")
    print("  Also,verify the full path to the folder containing the Parquet
                                          files.")
    df=pd.DataFrame()
else:
    df=pd.concat(all_dfs,ignore_index=True)
    print(f’\n---Data Loading Complete---’)
    print(f’All datasets combined.Total rows:{len(df)},Total columns:{len(df.
                                          columns)}’)
```

10.2 Data cleaning & pre-processing

Python code for handling infinite values , checking whether dataset contains missing values.

```
print("---Starting Data Cleaning: Handling Infinite Values---")
df.replace([np.inf,-np.inf],np.nan,inplace=True)
nan_after_inf_check=df.isnull().sum()
if nan_after_inf_check.sum()>0:
    print("NaN values introduced after replacing infinities:")
    print(nan-after_inf_check[nan_after_inf_check>0])
    print("\nDeciding how to handle these NaNs:")
    df.fillna(0,inplace=True)
    print("Filled all introduced NaNs with 0.")
else:
    print("No infinite values found and no NaNs introduces.")
print("-"*50)
pd.reset_option(’display.max_columns’)
pd.reset_option(’display.max_rows’)
print(’\n4. Missing values per column (df.isnull().sum()):’)
print(df.isnull().sum())
print("-"*50)
```

Python code for converting categorical network features into a numerical format, suitable for machine learning algorithms.

```
#label encoding
print("---Preprocesing: Encoding Target ’Label’ Column---")
if ’label’ in df.columns and df[’label’].dtype==’object’:
    le=LabelEncoder()
    df[’label_encoded’]=le.fit_transform(df[’label’])
    print(f"Original Lable Value Counts:\n{df[’label’].value_counts()}")
    print(f"\nEncoded Label Value Counts:\n{df[’label_encoded’].value_counts()
                                              }")
    print("\nMapping of original labels to encoded numbers:")
    for i,label in enumerate(le.classes_):
        print(f"{label}->{i}")
    SAVE_DIR = ’models’
    if not os.path.exists(SAVE_DIR):
        os.makedirs(SAVE_DIR)
        print(f"Created directory: {SAVE_DIR}")

    LABEL_ENCODER_FILENAME = ’label_encoder.joblib’
    LABEL_ENCODER_PATH = os.path.join(SAVE_DIR, LABEL_ENCODER_FILENAME)

    # Save the fitted LabelEncoder object
    joblib.dump(le, LABEL_ENCODER_PATH)

    print(f"LabelEncoder saved successfully to: {LABEL_ENCODER_PATH}")

elif "label" in df.columns and df[’label’].dtype!=’object’:
    print("’Label’ column is already numerical.Skipping Label Encoding.")
    df[’label_encoded’]=df[’label’]
else:
     print("Warning: ’Label’ column not found for encoding. Please verify the
                                               name of your target column.")
print("-"*50)
print("\n--- Skipping One-Hot Encoding for Feature Columns ---")
print("All feature columns are already numerical (not ’object’ dtype), so no
                                         One-Hot Encoding is needed.")
print("-" * 50)
```

Python code for scaling numerical network features to a consistent range,ensuring balanced influence during model training.Âă

```
print("\n---Starting Preprocessing: Scaling Numerical Features---")
numerical_features=df.select_dtypes(include=[’int8’,’int16’,’int32’,’int64’,’
                                         float32’,’float64’])
columns_to_exclude_from_scaling=[’label’,
    ’label_encoded’,
    ’protocol’,
    ’fwd_psh_flags’,
    ’bwd_psh_flags’,
    ’fwd_urg_flags’,
    ’bwd_urg_flags’,
    ’fin_flag_count’,
    ’syn_flag_count’,
    ’rst_flag_count’,
    ’psh_flag_count’,
    ’ack_flag_count’,
    ’urg_flag_count’,
    ’cwe_flag_count’,
    ’ece_flag_count’,
    ’downup_ratio’,
    ’fwd_avg_bytesbulk’,
    ’fwd_avg_packetsbulk’,
    ’fwd_avg_bulk_rate’,
    ’bwd_avg_bytesbulk’,
    ’bwd_avg_packetsbulk’,
    ’bwd_avg_bulk_rate’]
numerical_features_to_scale=[col for col in numerical_features if col not in
                                         columns_to_exclude_from_scaling]
if not numerical_features_to_scale:
    print("No numerical features found for scaling (after excluding labels/IDs
                                             ).")
else:
    print(f"\nScaling the following numerical features: {
                                             numerical_features_to_scale[:5]}
                                             ... (showing first 5 of {len(
                                             numerical_features_to_scale)}
                                             columns)")
    scaler=StandardScaler()
    df[numerical_features_to_scale]=scaler.fit_transform(df[
                                             numerical_features_to_scale])
    SAVE_DIR = ’models’
    if not os.path.exists(SAVE_DIR):
        os.makedirs(SAVE_DIR)
        print(f"Created directory: {SAVE_DIR}")
    SCALER_FILENAME = ’scaler.joblib’
    SCALER_PATH = os.path.join(SAVE_DIR, SCALER_FILENAME)
    joblib.dump(scaler, SCALER_PATH)
    print(f"Scaler saved successfully to: {SCALER_PATH}")
    print("\nFirst 5 rows of DataFrame after scaling (check numerical features
                                             ):")
    pd.set_option(’display.max_columns’, None)
    print(df.head())
    pd.reset_option(’display.max_columns’)
    print(f"\nDataFrame shape after scaling: {df.shape}")
```

10.3 Dataset preperation

Splitting the dataset into training and testing sets.It helps to avoid overfitting and to accurately evaluate your model.

```
#splitting data for training and test

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state
                                          =42,stratify=y)
```

10.4 Model Training& Evaluation

A Random Forest Classifier was trained on the prepared NIDS dataset, which was preprocessed using SMOTE to address class imbalance and improve detection of less frequent attack types.

```
print("\n--- Training RandomForestClassifier on SMOTE-resampled data ---")


rf_model_smote = RandomForestClassifier(
    n_estimators=100,        # Number of trees (can be increased for more
                                              performance)
    random_state=42,         # For reproducibility
    n_jobs=-1,               # Use all available CPU cores for faster training
    verbose=1                # Show training progress
)

print(f"Training data shape (resampled): {X_train_resampled.shape}")
print(f"Training target shape (resampled): {y_train_resampled.shape}")

rf_model_smote.fit(X_train_resampled, y_train_resampled)
print("\nRandomForestClassifier training on SMOTE-resampled data complete!")


print("\n--- Evaluating the Model (trained with SMOTE) on the ORIGINAL Test
                                          Set (Multi-class) ---")

y_pred_smote = rf_model_smote.predict(X_test)

accuracy_smote = accuracy_score(y_test, y_pred_smote)
print(f"Accuracy on Test Set (after SMOTE training): {accuracy_smote:.4f}")

print("\nClassification Report (Multi-class, after SMOTE training):")

target_names_multi = [encoded_to_label[i] for i in sorted(encoded_to_label.
                                          keys())]
print(classification_report(y_test, y_pred_smote, target_names=
                                          target_names_multi, zero_division=0)
                                          ) # Added zero_division=0 to prevent
                                           warnings for classes with 0
                                          precision/recall
```

```
print("\nConfusion Matrix (Multi-class, after SMOTE training):")
print(confusion_matrix(y_test, y_pred_smote))

# ---  Evaluate the model on the ORIGINAL, UNTOUCHED test set (Binary
                                          Malicious vs. Not Malicious) ---
print("\n--- Binary (Malicious vs. Not Malicious) Classification Report (after
                                           SMOTE training) ---")

# Convert true labels (y_test) to binary (using the original y_test)
y_test_binary_smote_eval = np.where(y_test == NORMAL_ENCODED_VALUE, 0, 1)

# Convert the NEW predicted labels (y_pred_smote) to binary
y_pred_binary_smote_eval = np.where(y_pred_smote == NORMAL_ENCODED_VALUE, 0, 1
                                          )

print(classification_report(y_test_binary_smote_eval, y_pred_binary_smote_eval
                                          , target_names=[’Not Malicious (0)’,
                                           ’Malicious (1)’], zero_division=0))

print("\n--- Binary Confusion Matrix (after SMOTE training) ---")
print(confusion_matrix(y_test_binary_smote_eval, y_pred_binary_smote_eval))


print("\n--- Investigating FALSE NEGATIVES (Actual Malicious, Predicted Not
                                          Malicious) after SMOTE training ---"
                                          )

false_negatives_indices_smote = np.where((y_test_binary_smote_eval == 1) & (
                                          y_pred_binary_smote_eval == 0))[0]

if len(false_negatives_indices_smote) > 0:
    print(f"\nTotal FALSE NEGATIVES (after SMOTE training): {len(
                                              false_negatives_indices_smote)}"
                                              )

    # Get the ACTUAL multi-class labels for these False Negatives
    # Use .iloc[] for robust indexing with Pandas Series
    actual_labels_for_fns_smote = y_test.iloc[false_negatives_indices_smote]

    # Convert encoded actual labels back to human-readable
    actual_fn_labels_smote = [encoded_to_label[label] for label in
                                              actual_labels_for_fns_smote]

    # Analyze the distribution of actual types of missed attacks
    missed_attack_counts_smote = pd.Series(actual_fn_labels_smote).
                                              value_counts()
    print("\nDistribution of Missed Attack Types (False Negatives) after SMOTE
                                               training:")
    print(missed_attack_counts_smote)
else:
    print("No FALSE NEGATIVES found after SMOTE training. Model is perfect at
                                              not missing attacks!")
```

10.5 Model Deployment and Application Integration

Loading Model And Preprocessing Tools

```python
model = None
scaler = None
label_encoder = None
label_encoder_classes_list = None # To store classes as a list for robust decoding

try:
    model = joblib.load(MODEL_PATH)
    print("Model loaded successfully.")
except FileNotFoundError:
    print(f"Error loading model file: {MODEL_PATH} not found. Make sure the path is correct and the model provider has placed the file.", file=sys.stderr)
    sys.exit(1) # Critical error, exit
except Exception as e:
    print(f"An unexpected error occurred loading model: {e}", file=sys.stderr)
    sys.exit(1) # Critical error, exit

try:
    scaler = joblib.load(SCALER_PATH)
    print("Scaler loaded successfully.")
except FileNotFoundError:
    print(f"Warning: Scaler file: {SCALER_PATH} not found. Ensure your model was not trained on scaled data, or provide the scaler.", file=sys.stderr)
    # This is a warning, not an exit, as some models might work without scaling, but accuracy will suffer.
except Exception as e:
    print(f"An unexpected error occurred loading scaler: {e}", file=sys.stderr)

try:
    label_encoder = joblib.load(ENCODER_PATH)
    label_encoder_classes_list = list(label_encoder.classes_) # Convert to list
    print("Label encoder loaded successfully.")
except FileNotFoundError:
    print(f"Warning: Label encoder file: {ENCODER_PATH} not found. Predictions will be numerical.", file=sys.stderr)
except Exception as e:
    print(f"An unexpected error occurred loading label encoder: {e}", file=sys.stderr)
```

10.5.1 Data Preprocessing: Handled by preprocess_data()

The `preprocess_data()` function takes the raw CSV data and, after all its cleaning, renaming, feature selection, and scaling steps, it returns a df_for_prediction which is a processed DataFrame containing only the features the model expects, in the correct format.

10.5.2 Prediction Logic: Handled by predict_intrusion()

```
def predict_intrusion(input_df_raw):
    """
    Takes raw input DataFrame, preprocesses it, and returns predictions.
    """
    if model is None:
        return pd.Series(["Error: Model not loaded."], index=input_df_raw.
                                                  index)

    # 1. Preprocess the input data using the *same* steps and fitted scaler/
                                              encoder
    try:
        processed_df = preprocess_data(input_df_raw, scaler)

        # 2. Make predictions
        predictions = model.predict(processed_df)

        # 3. If your target was label encoded, decode it back for human
                                                  readability
        if label_encoder_classes_list: # Use the list for robust decoding
            decoded_predictions = []
            for pred_val in predictions:
                pred_val_int = int(pred_val) # Ensure integer for indexing
                if 0 <= pred_val_int < len(label_encoder_classes_list):
                    decoded_predictions.append(label_encoder_classes_list[
                                                              pred_val_int])
                else:
                    decoded_predictions.append(f"UNKNOWN_PREDICTED_LABEL_{
                                                              pred_val_int}")
        else:
            decoded_predictions = predictions

        return pd.Series(decoded_predictions, index=input_df_raw.index)

    except ValueError as ve:
        return pd.Series([f"Preprocessing Error: {ve}"], index=input_df_raw.
                                                  index)
    except RuntimeError as re:
        return pd.Series([f"Preprocessing/Scaling Error: {re}"], index=
                                                  input_df_raw.index)
    except Exception as e:
        return pd.Series([f"Prediction Error: {e}"], index=input_df_raw.index)
```